***OK COMPOSER*:**

SYMBOLIC MUSIC GENERATION WITH RECURRENT NEURAL NETWORKS

A thesis submitted to attain the degree of
Master of Science at University of Copenhagen

M. Sc. IT & Cognition

Author:
Cristian Mitroi

Supervisor:
Manex Aguirrezabal Zabaleta

June, 2019

60 pages, 143986 characters

A massive thank you to my supervisor, Manex Aguirrezabal Zabaleta, for all the guidance he offered through the process of research, experimenting, and documentation.

A debt of gratitude to Hendrik Purwins and the Sound and Music Computing group at Aalborg University, for the fruitful and stimulating talks on the topics of music and technology.

Dedicated to my supportive partner, friends, and family: you were always there for me, listening to my nerdy excitement for the topic of music generation.

Contents

Appendix A: Introduction to Neural Networks

Appendix B: Music theory

Abstract

The current work aims to address the task of symbolic music generation with recurrent neural networks. I provide an overview of the state of the art techniques in the field, with a discussion of how these techniques have been employed on this specific task. I then conduct experiments on how different factors affect learning. I test for dataset properties (stylistic homogeneity, encoding format) and sequential mechanisms (attention mechanism, bidirectionality). In evaluating the models I employ an objective methodology that compares the generated samples with the training sets on multiple musical domain statistics. I also measure the degree of plagiarism in each of the generated sets, as compared with the training samples. I provide a discussion on how and why the mechanisms impact symbolic music generation. I then perform a subjective analysis of the quality of the samples generated. Finally, I conduct a user study comparing the generated sets with the training sets. The most important conclusions are: the attention mechanism commonly employed in machine translation does improve learning in symbolic music generation in this experiment; dataset homogeneity has a prominent impact on learning; bidirectional LSTMs do offer an improvement; the model trained on the pianoroll format has a slightly more varied rhythmic vocabulary than the *melody* counterpart.

1 Introduction

Music composition

Music composition can be seen as an entirely human endeavour, a very complex system of rules and conventions that yields a final aesthetically pleasing result for the ears. It is also a strictly subjective endeavour, as it is always a matter of taste what sort of music you enjoy. Good music is hard to describe, though scholars and musicians agree that there are in general some common rules of harmony, rhythm, and melody that form the basis for pleasing experiences [1] [2] [3].

The quest to have a computer compose *good* music has been a long one. The earliest attempts involved manually-crafted Markovian tables. More recently, with the arrival of deep learning methods and the decreasing price of hardware (especially graphical processing units), we see widespread interest in the topic. There are plenty of amateur web blog posts detailing how to program and train such a system.[1]

With the advent of artificial neural networks more sophisticated approaches arose, like the ones by [4] and [5]. More recently, the attempt has become a part of the mainstream trend, with Google even developing a music generation doodle for celebrating Johann Sebastian Bach's birthday ([6], based on [7]) and dedicating an entire team to the field of generating art with machine learning.[2] In fact, master composers like Bach seem to hold a certain fascination in the field, with his corpus of chorales being used with great success in multiple models ([8], [9], [10]). Overall, there has been a great increase in music generation approaches and architectures over the last years [11].

I make the distinction between audio music generation and symbolic music generation. All of the efforts in this paper and the project itself deal exclusively with symbolic music generation. This means that the music itself is not in the audio format that we experience when listening to it. Rather, it is encoded into some digital format that can be processed by computers. By far, the most common format for musical transcription in the digital age is MIDI (Musical Instrument Digital Interface). I provide an overview of the format in the MIDI section in the Appendix. It is the digital equivalent of sheet music. While there are indeed efforts to produce music directly at the sound level [12], I choose to work on the symbolic level. In symbolic format musical notes are represented as a sequence, thus allowing us to employ probabilistic models (like an RNN) to model this problem. In my paper, I employ the more advanced LSTM cell for my sequential learning. The state of the art in this regard is dealt with in State of the art.

[1] www.deeplearning.net/tutorial/rnnrbm.html
www.hexahedria.com/2015/08/03/composing-music-with-recurrent-neural-networks
www.wise.io/tech/asking-rnn-and-ltsm-what-would-mozart-write
https://towardsdatascience.com/deep-learning-with-tensorflow-part-3-music-and-text-generation-8a3fbfdc5e9b

[2] https://magenta.tensorflow.org/

Problem statement

My hypothesis for this project is related to the effects of (1) **dataset properties** and (2) **sequential learning mechanisms** on symbolic music generation in deep learning.

The first part deals with how neural networks learning is impacted by the choice of dataset. In the case of symbolic music generation, the model is trained on a dataset of samples[3] and then the user employs a sampling method based on a choice of a seed sequence to create new sequences from the probability distribution. I argue that the choice of dataset, especially in music generation, has a major impact on the quality of the results. This is even more important in music generation, because of the problem of musical style, or genre. Datasets with different distributions of musical styles will impact learning in different ways.

In this project, I will compare the effects of using a stylistically homogenous dataset with a stylistically heterogeneous dataset. More precisely, I will compare a model trained on a dataset consisting of solely folk music (traditional Irish tunes) with a model trained on a dataset consisting of the same number songs, but of different genres (pop, jazz, rock, classical etc.). The comparison will be done in terms of a quantitative analysis, based on musically-informed features [13], but also in terms of a subjective analysis of the samples. It will also consist of a measure representing originality and plagiarism of the model, using a state of the art algorithm [14] [15].

In most research in the field, researchers tend to use either a single style of music (e.g. classical in [8] [10]) or an amalgamation of styles (in [16], [17]). It is important however to compare results between training on a homogenous versus a heterogeneous dataset. I think that such a comparison can yield relevant insight into how a symbolic music generation model might perform depending on the dataset.

Also on the topic of dataset properties, I investigate the difference between two encoding formats, the pianoroll and the *melody* format. The former has been employed by numerous projects in the field, being the de-facto choice, while the latter has been recently introduced by the Magenta team in [17]. I provide more details on these in the section on formats in the dataset chapter. I compare how the choice between these can affect the performance of the model.

The second element of my problem statement is a discussion on how different deep learning techniques for time series data can affect learning. Specifically, I analyze how the attention mechanism in sequential models, as defined in [18], [19] and [20], can be used to improve performance of sequential Long Short-Term Memory models. The mechanism provides the model with a system for weighing the relevance of each of the timesteps in the input chain. As my focus is music generation, I argue that attention allows the system to learn which parts of the musical piece are central in predicting the next time step element. This can be considered an analogy to how a composer will consider not just the last note at $t-1$, but perhaps an entire 2-bar sequence, when choosing the current element (at timestep t).

Another sequential deep learning technique is the bidirectional wrapper [21] for the LSTM layers, that allows the model to consider both time directions, forward and backward, during training time. This allows for the model to learn the context of the piece better, as it considers it both in chronological and reverse chronological order. This can be considered an analogy to how a composer will plan their melodic progression in terms of a tonic key, that they will eventually return to at the end of the piece[4].

Lastly, I would like to mention that this project deals strictly with monophonic symbolic melody gener-

[3] By "samples" I am referring to samples of MIDI music. These are not audio samples. Whenever I am referring to music in the context of the current project, it will be about MIDI/symbolic music.

[4] I elaborate on music theory in Appendix B

ation. This means that all tracks in the training set and in the generated set consist of a single melody line. There are no passages where there will be more than one note playing at a given time. My system thus does not cover harmony (chord generation) or polyphony (counterpoint generation etc.).

From the musical perspective, the melody is the piece in the song that has the most lasting impact on the listener's memory. The scholarship in musical theory and music composition is that "the melody is usually the most memorable aspect of a song, the one the listener remembers and is able to perform" [22, p. 2]. Thus, it is the melody that holds a central position in the musical experience of the listener. I thus decided to focus my efforts on this. Another advantage of a monophonic approach is that it allows me to employ a state of the art algorithm to compare melodic contours to determine plagiarism and originality [14] [15]. The melodic contour, as defined in [22, p. 40], is defined by a series of characteristics that can be modelled by computer algorithms.

The current work can also be framed within the four challenges facing music generation with machine learning, as presented by Briot and Pachet in [23]. More specifically, the attention mechanism and bidirectionality attempt to address the issue of **structure**, while the plagiarism evaluation is related to the issue of **originality**. I will discuss these four categories further in the paper.

Methodology

I produce four experiments where I change a certain aspect of the architecture and compare the difference in both a quantitative and a subjective analysis. I also provide an analysis of the originality (plagiarism) of the systems in the experiments. Finally, I produce a qualitative assessment of the samples produced by one of the systems by using a user study.

I will provide a thorough objective evaluation of the generated results, with comparison to the training dataset. I believe this is essential to gaining insight into the learning process, especially with regards to non-subjective interpretations and statistical analyses. I utilize the methods outlined by Yang and Lerch in [13] that are designed with domain knowledge of music theory. This allows for capturing useful statistical knowledge of the variations in the datasets. The metrics obtained from these allow for formative comparison of models with relation to the training set. It forms a measurable way of representing how similar the samples generated by the model are when compared with the training sequences.

Another key element in my approach is also related to the evaluation. A key challenge in the music generation field is the element of originality, as compared with plagiarism [23]. This can be seen as the model simply learning to reproduce the musical sequences that it has been exposed to, rather than producing new, original material. For this, I propose to employ a state of the art algorithm for melody similarity [14] in comparing generated sequences to the dataset of training examples. This provides a measure for similarity and plagiarism. The laws for musical plagiarism are complex and intricate, with several important cases being discussed in [24]. However, I think that a first step in the process of legitimizing music composed by computers is to have a methodology of plagiarism detection.

In the subjective evaluation of each of the systems, I attempt to employ it as a musician would. I compose a short original piece of melody and use it as a seed sequence for the systems. I then evaluate how the generated samples fit and continue that melody.

Motivation

My motivation for this project is twofold: firstly, as an amateur musician myself, I see technology as being a tool for creative expression. Thus, this system should not be perceived as a replacement of musicians. It does not seek to completely replace the human element in the process of music creation. Rather, it seeks to aid the human in generating new ideas based on a set of parameters, given by the human. Finally, it would, of course, be the human that chooses which of the generated pieces fit their needs best, and in which ways they should be altered and processed. I present two recent examples of success in this merging of the computer and the human: the 2017 Titan V launch, where an AI-composed symphony was conducted and played by a human orchestra;[5] and the music album "Hello World,"[6] composed by the French artist Benoit Carré with the help of an AI powered by the system described in [25]. These show what can be achieved when human creativity is boosted by the power of Deep Learning systems. These allow for faster idea generation and iteration. Music generation is also part of a greater interest in the field of AI for the creative arts, with other endeavours in the field of text generation [26], poetry generation [27], and painting generation [28].

Secondly, I approach this problem from the curiosity of a technical-minded researcher. We can see music generation as a problem to *solve*, similar to modelling stock prices fluctuations or the written language in NLP. What sort of patterns are inherent in the language and fluctuations of music? What is the optimal encoding format to capture these melodic patterns? This motivation is more related to my goal of comparing dataset formats and dataset style homogeneity impact on the quality of the model. The attention mechanism also allows for insightful visualizations, as I will show later on. My effort becomes part of the greater engineering effort for modelling music, as outlined in the survey in [11].

Structure

The paper is structured in the following way:

1. chapter two covers the background. I first discuss the general background of deep learning for sequential data. I present each of the techniques as having arisen out of a need to solve more complex problems and to overcome limitations of a specific method (e.g. LSTMs attempt to solve the "vanishing gradient" problem in RNNs). I then present the field of symbolic music generation. I consider the papers and projects that have been most influential to my work. I also frame my work within the greater effort to "solve" the task of symbolic music generation, describing how it solves some of the challenges faced by the field, as outlined in [23] by Briot and Pachet. I also discuss the problem of plagiarism detection in symbolic music.
2. chapter three covers the methods employed in my project. I discuss and argue for how each of the deep learning architecture methods fit my problem theoretically, in light of other work and analogies to music theory. I also discuss the methodologies for the quantitative analysis, plagiarism analysis, subjective analysis, and user study.
3. chapter four is an overview of the datasets I use in this project. I detail their origin, provide a statistical comparison of their features. I then discuss the two encoding formats that I will use in my experiments. I conclude with the preprocessing procedure.
4. chapter five begins with a re-iteration of the problem statement hypotheses, followed by a thorough discussion of each of the experiments. I review in detail the architecture of the models used. I then provide various visualizations and informative statistics to perform an objective analysis. I

[5] https://blogs.nvidia.com/blog/2017/12/07/titan-v-launch/
[6] https://www.helloworldalbum.net/about-hello-world/

also contribute a table of "plagiarism scores" for the models. I conclude each experiment with a subjective analysis of how the models perform when used to produce continuations of an original melody. The chapter ends with a review of the user study and its results.

2 | Background & State of the Art

In this chapter, I provide an in-depth and thorough overview of the background research relevant to my project. I present salient theories and framework from deep learning and symbolic music generation using machine learning methods. I discuss the major breakthroughs and significant topics that are essential to my research.

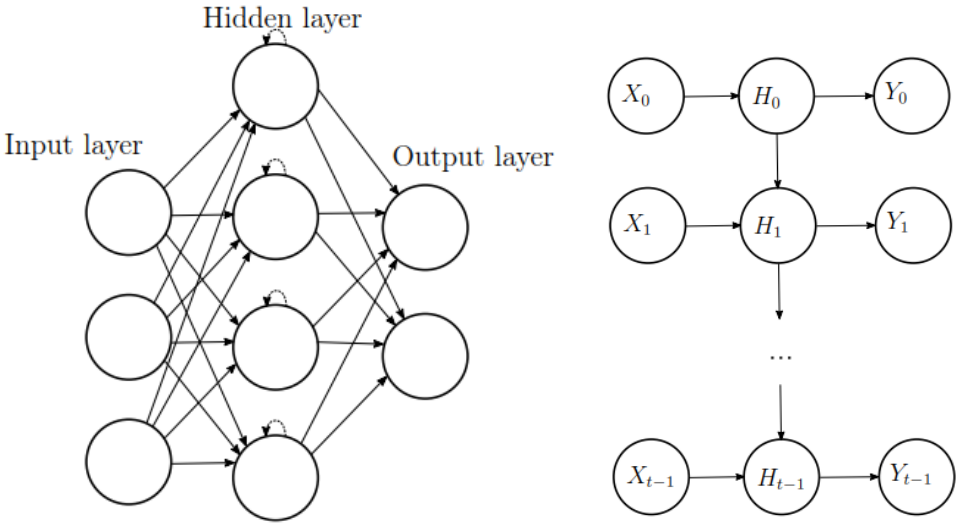
In Appendix A I provide an overview of machine learning and neural networks in general. I discuss backpropagation and different methods of optimization.

In Appendix B I offer a short introduction to key musical theory concepts necessary to understanding my decisions in this paper.

Recurrent Neural Networks

The assumption under which we work when we choose a feed-forward network is that the data is identically and independently distributed. While this is the case for some types of data sets, it does not hold for the case for temporal data. Examples of time-based data are text (for e.g. sentiment prediction), audio signals (for e.g. voice to text systems), and symbolic music (for e.g. symbolic music generation or classification). These data types exhibit complex sequential patterns that are completely invisible to a Multi-Layer Perceptron.

A Recurrent Neural Network (RNN) is a type of neural network that allows for these temporal patterns to be learned [29]. It achieves this by a method of a feedback loop, where the system maintains an internal state of the previous step in the learning process instead of dropping it entirely, as the feed-forward networks do. In other words, the output at a given timestep also depends on the previous timesteps. In fig. 2.1a we can see an example of an MLP, but with the added temporal feedback loop represented by the dotted arrows coming out of hidden layer cells. This graphical representation of a RNN is called the *folded* representation. This is because the feedback loop is represented as feeding into the same figure. Comparing this with the *unfolded* representations in fig. 2.1b, we notice that the latter exhibits a clearer graphical representation of the temporal nature of the RNN. The internal state H gets updated at every time step. X is the input at the specific time step while Y is the output.



(a) RNN with feedback loop

(b) Unfolded RNN

Figure 2.1: RNN Architecture

Formally, a sequence of vectors x is processed as follows[1]:

$$h_t = \sigma(Ux_t + WH_{t-1})$$

$$y_t = \text{softmax}(VH_t)$$

Where U is the weight vector for the hidden layer, V is the weight vector for the output layer, W is that weight at a different time step, x_t is the input vector, y_t is the prediction at the time step t, and h_t is the hidden state at the time step

RNNs are not the only sequential models for learning. There are also Markov Chains. However, as pointed out in [30, p. 1], "the advantage of recurrent networks is that they are not restricted to Markovian prediction". This means that RNNs are more sophisticated systems and can develop complex temporal patterns. In a similar way, Vitelli and Nayebi observe that, with Markov approaches, the music produced does not exhibit thematic structure and is overly repetitive: "the resulting musical pieces usually consist of repetitive sequences and lack thematic structures that are common in most musical works" [31, p. 1].

RNN tasks

RNNs can be employed in a variety of tasks, depending on the nature of the input and output. While the feed-forward neural networks are limited to inputs and output that are a single vector, RNNs do not suffer from that limitation. Inputs and/or outputs can be both a single vector but also a sequence of vectors. Andrej Karpathy [32] uses the nomenclature of "many-to-one", "one-to-many", and "many-to-many". He also provides instances of each of these. In this categorization, an MLP is a one-to-one network.

- one-to-many: the input is a single vector, but the output is a sequence. An example of this is image captioning systems, where we have a single image as input and multiple words forming a sentence as output;

[1] Based on the equations from https://medium.com/deep-math-machine-learning-ai/chapter-10-deepnlp-recurrent-neural-networks-with-math-c4a6846a50a2

- many-to-one: this is the opposite of the previous type. We have multiple vectors forming a sequence as input, and we have one single vector as output. A case of this would be sentiment prediction, where we have a sequence of words as input and a single one-hot encoding vector of the predicted sentiment. This is also where my model fits, but with the added note that I use a sampling method that allows me to feed the system its own output in order to produce a longer sequence. More on this in the next chapter;
- many-to-many: in this scenario both the inputs and the outputs are sequences of vectors. There are two further subtypes here. The synchronous many-to-many produces output at the same time as the input is being fed. An example of this would be frame classification in video data. The asynchronous many-to-many, on the other hand, only produces the output after the entire sequence of input has been processed. An example of this is machine translation, where the system only produces the translation after the input sentence has been processed.

Backpropagation through time & RNN limitations

RNNs also employ the MLP gradient descent and backpropagation algorithms for learning the weights of the network. However, RNNs adapt it to suit the temporal nature of its learning. The approach is now called Back-propagation through Time (BPTT). This is achieved by first unrolling the network (see previously discussed fig. 2.1b) and then propagating the weight $\triangle$. In this form, a RNN is a MLP with the same number of hidden layers as the RNN has time steps [33, p. 342].

The main problem with RNNs is called the vanishing gradient problem. This happens when the $\triangle$ signal for updating weight shrinks exponentially as it is being backpropagated through the network's layers. The opposite could happen as well: the signal could grow exponentially and get out of bounds. This latter problem is referred to as the exploding gradient problem [33] [34]. To combat these limitations we examine Long Short-Term Memory networks. These offer a mechanism to control the flow of the gradient in a more granular manner.

Long Short-Term Memory Networks

LSTMs are an adaptation of RNNs systems. They were part of a set of different remedies to the vanishing/exploding gradient problem, as discussed in [34]. LSTMs yielded the most promising results, solving "complex, artificial long time lag tasks that have never been solved by previous recurrent network algorithms".[34, p. 11].

Initially proposed by [35], such a network can maintain proper handle of over 1000 time sequences [36]. Using different gate mechanisms, LSTMs allow for advanced gradient flow optimization. They can also learn to maintain or discard memory contents throughout the training process. On the other hand, RNN units "overwrite their memory at each time-step" [36, p. 17]. Basically, the problem inherent in RNNs is their limited temporal distance memory: "the memory capacity of simple backpropagation through time is practically limited" [30, p. 1]. LSTMs can be seen as being able to protect their weight gradient from unwanted disruption [37].

We use the following notation to define the LSTM learning process:

- x_t is the input vector at timestep t;
- f_t is the "forget" gate state at same timestep;
- o_t is the "output" gate;
- i_t is the "input" gate;

- c_t is the memory contents of said cell. This is computed based on the respective data point and internal weights state;
- h_t, finally, is the output vector one LSTM cell at timestep t;
- σ is the sigmoid function;

Then, formally, LSTM networks can be defined with the following equations:

$$h_t = o_t \tanh(c_t)$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t)$$

$$c_t = f_t c_{t-1} + i\tilde{c}_t$$

$$\tilde{c_t} = \tanh(W_{xc}x_t + W_{hc}h_{t-1})$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1})$$

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1})$$

The notation W_{xo} represents the weights meant for the input x and the output gate o.

From these equations, we can observe that indeed the output gate controls the output of the cell. Similarly, the input gate and forget gate control what updates the memory cell state.

We can see these equations represented in graphical form in fig. 2.2. The equations are based on [36] and [38].

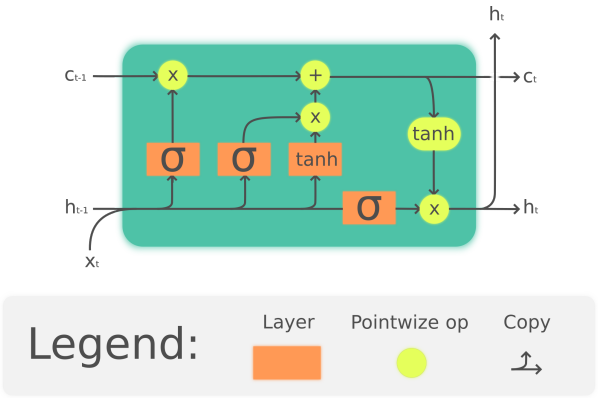


Figure 2.2: LSTM cell[2]

Bidirectionality

Standard LSTMs suffer the limitation that they cannot consider the upcoming context during training. Bidirectionality allows an LSTM to consider thus both positive and negative time direction when predicting a certain time step [21]. This feature is notably useful when the context of the input is crucial: handwriting to text, or part-of-speech tagging, or symbolic music generation. It is implemented an additional layer of LSTM cells, with the difference that this layer considers the input in a negative time direction, starting with the last time step and ending with the first.

[2]Image by Guillaume Chevalier, from https://commons.wikimedia.org/wiki/File:The_LSTM_cell.png. Creative Commons Attribution 4.0 International license.

Formally, the bidirectional wrapper is defined as follows[3]:

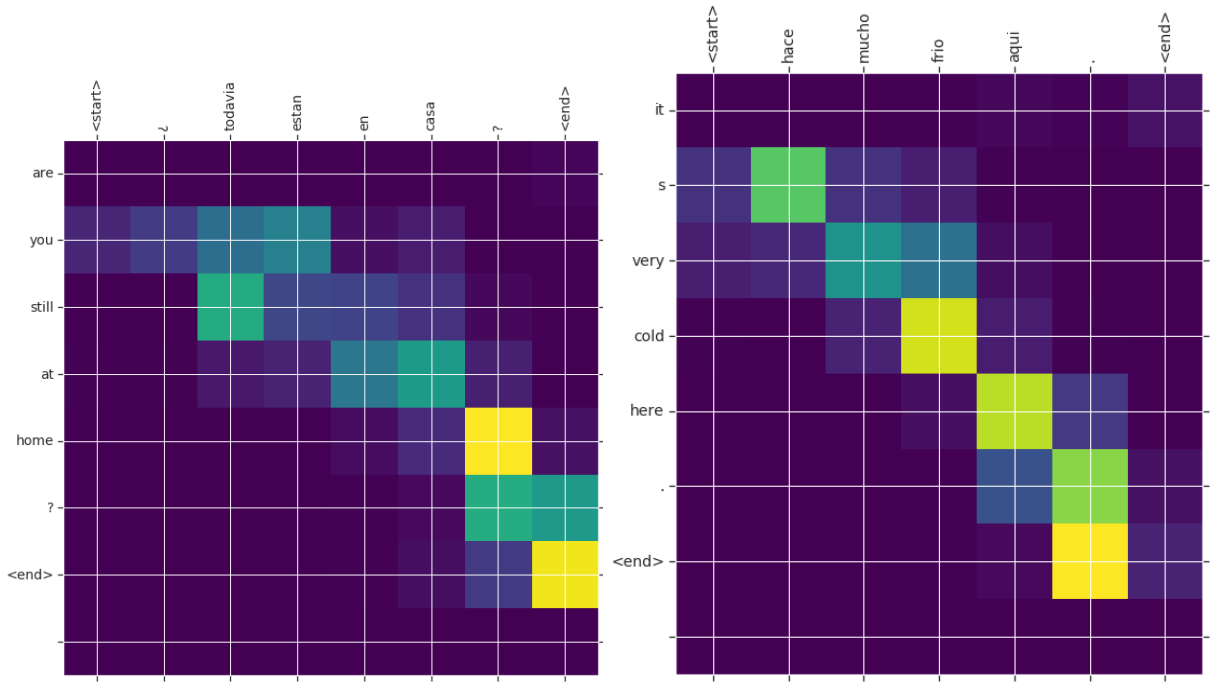
$$y_t = \mathrm{g}(W_y[\overleftarrow{x_t}, \overrightarrow{x_t} + b_y])$$

Where y_t is the output at that time step, g is some activation function, W_y are the layer weights, $[a, b]$ is the concatenation operation, x_t is the input, $\overrightarrow{x_t}$ is the input in the positive time direction, $\overleftarrow{x_t}$ is the input in the negative time direction, and b_y is the bias. We notice that a bidirectional wrapper simply concatenates the results from parsing the input in the two time directions. There are also other operations that could be applied, but concatenation is the standard [39, p. 52].

Attention

Attention mechanism [18] [19] [20] models the way in which humans perceive information. We focus more on certain regions, e.g. of an image, and less so on the less relevant parts. Similarly, when reading a sentence, we focus on a different part of a sentence at a time in order to form a coherent representation of its meaning.

Attention was originally developed to help alleviate limitations in neural machine translation [19]. Here the network learns a weight matrix for each of the time steps that it considers when predicting a next time step. The weights can then be visualized in a heatmap (see fig. 2.3a[4]).



(a) Alignment matrix of "todavia estan en casa ?" (Spanish) and its English translation "are you still at home ?"

(b) Alignment matrix of "hace mucho frio aqui" (Spanish) and its English translation "it s very cold here"

Figure 2.3: Attention alignment matrix examples

We can see that the heatmap confirms that attention learns to match the correct words across languages:

[3] Based on the 'Machine Learning' course by Andrew Ng, from here: https://www.coursera.org/lecture/nlp-sequence-models/bidirectional-rnn-fyXnn.

[4] Images from https://www.tensorflow.org/alpha/tutorials/text/nmt_with_attention. Creative Commons Attribution 4.0 License.

'frio' with 'cold'; 'casa' with 'home'.

There are multiple forms of attention, as discussed in [40]. As used in this paper, it is modeled after the work done in [18]. Here the authors use a single parameter per channel. Instead of a many-to-many architecture, we have a many-to-one. Formally, it is defined as:

$$e_t = h_t w_a$$
$$a_t = \frac{\exp(e_t)}{\sum_{i=1}^{T} \exp(e_i)}$$
$$v = \sum_{i=1}^{T} a_i h_i$$

Where h_t is a word at timestep t and w_a is the attention layer weights. We obtain a_t, which is the attention scores for timestep t, by weighing the representations of the words, and then normalizing. This results in a probability distribution over the tokens. The attention weights are then reused for generating every new time step of music, while also being adjusted during the learning process.

Attention allows the model to "decide" the relevance of each music token for the prediction of the upcoming next token. For example, tokens such as the notes C3, A3, G3 are strong indicators that the piece is in C Major and that the next note should also be in that key.

Symbolic music generation

In this section, I will provide a thorough overview and discussion of the research in symbolic music generation. I will analyze proposals made throughout the field, dissect in what way they were innovative, and in what way they are an influence on my own current work. This will not be exhaustive, I will focus on those works that were influential. I will mention other related work, for context and history.

Advances in deep learning methods have allowed us to yield promising results from probabilistic models trained on examples of symbolic music. These models develop an ability to generate music based on rules learned from studying large collections of existing music [30]. Before this, systems for making music were based on programming composition rules beforehand, based on expert knowledge of the field

One of the largest survey papers in the field is done in [11] by Briot et al. This covers a wide spectrum of published research on the topic. They also discuss concepts related to music data encoding formats and music theory. They provide an in-depth comparison of different strategic approaches to music generation, and discuss different challenges faced by the field.

One of the first systems based on recurrent networks, CONCERT by Mozer, relied on estimating the chance of producing a given pitch by considering the previously played ones [4]. The input was encoded as a combination of note pitch, time duration, and chord harmony. The system was trained on melodies from Bach. This system was one of the historical landmarks in the field. By comparison, my system also works on estimating and predicting melodic content. However, I am also employing more sophisticated methods, such as bidirectionality, LSTMs, and the attention mechanism. Mozer was also one of the first to observe RNN limitations, noting its restriction with regards to global structure in music.

In the same paper, Mozer compares RNNs with basic Markovian transition tables. These sort of tables were the prevailing methods for music generation before the arrival of the more complex RNNs and LSTMs. This was a very simplistic approach. The more complex approaches to Markovian predictions

involved n-degree tables, where the next note was predicted based on the previous n notes. However, using this approach they encountered memory and complexity limitations. Systems such as these are also limited in terms of what they can learn from data. On the other hand, neural systems such as RNNs can adapt to any temporal patterns, be them local (within a given relatively short history of time steps) or global (within the greater context of a musical piece). The system learns to adapt to the patterns in the data, in a way in which it does not require a pre-programmed set of boundaries. It seeks to uncover the relevant aspects of the input sequence and memorize only those parts that are relevant within its context layer [4].

A similar pioneer piece of research was Eck's and Schmidhuber's work on using Long Short-Term Memory networks for generating blues melodies [37]. They argue that using a simple pianoroll encoding, as opposed to the manually-crafted features of CONCERT, will teach the network by inductive bias to lean towards harmonically related notes, as opposed to chromaticism. They also highlight the problem inherent in the pianoroll representation, the "note ending" problem [11, p. 24].[5] They train their system on sequences of 12-bar blues, a very popular musical form in blues and jazz. The purpose of their project was to test whether LSTMs could learn chord progressions and melody outlines, "and then use that structure to advantage when composing new songs" (p.6). Their results were positive, showing that indeed an LSTM can understand chord systems and fit its melody to their harmonies. They frame their work in relation to Mozer's CONCERT system and its limitation with regards to RNN backpropagation. In a similar manner, I am also employing the pianoroll encoding and LSTM networks. However, I am also comparing the pianoroll encoding with another format, the *melody* format.

In [41] Chung et. al employ Gated Recurrent Units to the task of polyphonic symbolic music generation. GRUs are an alternative to LSTMs, where the "input" and "forget" gate are merged together into one "reset" gate, making them less computationally demanding [42]. They compare negative log-likelihood probabilities and observe that the GRUs have results on par with LSTMs, even outperforming them in some cases. I do not employ GRUs, or compare them with LSTMs in the current work, though they are a worthy avenue of experimentation as an alternative to the LSTMs.

Another relevant system is Feynman Liang's *BachBot* [8]. This one is also based on a recurrent network model, using LSTM cells. By contrast to mine, this system is trained to produce polyphonic sequences in the style of Bach's chorales. The author employs a 3 layer-deep architecture for optimal balance between performance and overfitting. In preprocessing the data the author chooses to transpose all the songs to the same key of C Major. I perform a similar procedure in my preprocessing.

Bob Sturm et al. also employ LSTMs to generate melodies [43]. As my project, they are using a similar dataset of traditional folk melodies. In fact, the name of their system is *folk-rnn*. However, they use a textual (ABC) notation [44] that is different from the classic MIDI/pianoroll format. This notation contains information about time signature, key signature, different types of tokens for music events.

Their results are very promising, with both a quantitative and qualitative analysis. Indeed, they are one of the few projects that employ musically-inspired quantitative methods. They observe that the generated pieces show a strong sign of harmonic centre, adhering to a specific scale. They also make an essential observation with relation to the inherent bias in Western music culture: 1) the saliency of the personal style; 2) stylistic consistency within a given genre. These are taken into consideration during the subjective evaluation of the compositions. The authors state that the outputs appear to be "musically meaningful (repetition, variation, melodic contour, structure, progression, resolution)" [43, p. 13].

This paper was one of the central inspirations for the current work. I also use the folk melodies dataset

[5] I will discuss this at length in the dataset chapter.

and employ a quantitative analysis based on music theory. Stylistic homogeneity in a dataset is also one of the key points of one of my experiments. Like them, I then also perform a subjective analysis of the generated samples.

[30] offers a versatile LSTM-based system that generates pieces depending on genre and style, trained on classical composers. They use a Biaxial LSTM [45] to model both temporal and pitch dependencies and obtain results that conform to the established musical classification of the composers. The Biaxial LSTM works by considering both the time dimension (as in traditional LSTMs), but also the pitch dimension. The latter works by traversing the time steps in the pitch axis, and learning which notes play at the same time step(s). It basically forms a vertical approach, as a compliment to the existing horizontal approach. I do not employ this mechanism. I do use bidirectional LSTMs, which also provide another dimension to the learning process. In this case, bidirectionality adds a negative time dimension that provides a larger context to the system.

In [46] Choi et al. discuss using a word-RNN and char-RNN approach to generate chord progressions and drum tracks. They apply these models to datasets of jazz chord progressions and rock drum tracks. The word-RNN model outputs promising results on both datasets, while the char-RNN is only able to learn from chord progressions. They also allow for a α control parameter that adjusts the "technical virtuosity" of the generated pieces. A larger value (e.g. 1.5) yields tracks with more drum fills, while a lower value (e.g. < 1.0) will produce a track with nothing but kick, snare, and hi-hats. This is similar to the temperature t option that I employ in the sampling procedure (see sec. 3) They also employ a LSTM network, as I am doing in the current work. The main difference is in the dataset and its encoding. They train their model on chord progressions in text format (e.g. `_START_ C:maj C:maj G#:7 G#:7 _END_`) and drum tracks from Metallica in MIDI format.

In [47] Huang and Wu develop a LSTM model to generate classical music, trained on a corpus of Bach and other composers. They also experiment with different encodings of the MIDI tracks: one is a flattened approach of translating the time dimension into tokens (e.g. playing middle C for 480 ticks is composed out of the tokens `note-on-60-0` and `note-off-60-480`), and the other is the traditional pianoroll method.

They observe that the flattened approach does not produce as promising results, perhaps due to it not being able to learn about the vertical synchronous dimension of multiple tracks. They also embed the notes in an embedding layer, similar to the word2vec approach [48]. They observe meaningful clusters in a t-SNE visualization [49] : `note off`, `note on`, and different groupings by pitch range.

They also observe that the more aesthetically pleasing results arise from a model trained on only Bach data, as opposed to the entire dataset of classical music. This is similar to my hypothesis about dataset homogeneity and its effects on learning. In their case, their model was not able to learn the entire range of the dataset effectively due to the model's restricted size.

The current work is similar to Huang and Wo's paper, in that I also experiment with encoding methods and various datasets.

Another relevant model is the one proposed by the Magenta group in [17]. In this model, they compare a baseline of a simple LSTM model with an LSTM model enhanced with an attention mechanism. The attention mechanism is based on the work by [19]. In addition to the attention mechanism they also implement a 'lookback' token in their vocabulary, that informs the model that the current time step is a repetition of a previous step. Thus, it can better learn the long-term structure of music, where repetitions are essential for creating a pleasant experience. As pointed out by the author, most melodies in pop music rely on repetition of previous bars. They observe that the model trained with attention produces more coherent long-term structures in the melody.

This work was also influential on the current project. I also attempt to use the attention mechanism in an LSTM for music generation. However, the authors do not provide in-depth details on the evaluation or a discussion, as they only published the research as a blog post. I will provide a thorough evaluation and discussion in the results chapter.

Overall, it seems that sequential models, and, in particular, LSTMs are popular and efficient solutions in the field. They learn complex patterns in the music sequences in a way in which manually-defined Markov tables cannot capture. My work follows the research discussed in this section. I will employ these methods in my experiments and then discuss the results.

Challenges

A paper on the challenges in the domain of symbolic music generation is [23] by Briot and Pachet. They dissect the topic by categories, arguing for four main limitations in the field: control, structure, creativity, and interactivity. I will spend this section discussing these challenges, existing solutions, and how the present work fits into the four categories.

The first, **control**, deals with the user imposing musical constraints on the piece to be generated. These could be, for example: key, time signature, tempo, chord progression etc. These are hard to impose upon a neural network due to its very nature: they "are not designed to be controlled" [23, p. 2]. They do not offer a mechanism for controlling the inside workings of the black box. The authors present the case of the Anticipation-RNN [50]. This model achieves a level of control over the pieces by having certain notes play at specific upcoming time steps, thus conditioning the network.

I do not address this topic in the present work.

The second, **structure**, deals with long-term structure of a musical piece. In music theory, there are multiple forms that can be identified as global organization units. These could be the chorus, bridge or verse in pop; AABA or twelve-bar blues in jazz or blues; overture, finale, sonata in classical music. One model that addresses this problem is the MusicVAE [51] model. This system proposes a hierarchical VAE decoder, which first generates the encoded representation for each of the voices (a conductor) and finally generates each of the bars of music themselves with bottom-layer RNNs.

I argue that one of my hypotheses is part of the ongoing efforts in generating structure. Firstly, LSTMs address this problem by offering a mechanism of controlling the flow of the gradient, thus preventing it from "vanishing" or "exploding". The attention mechanism also allows the system to learn to focus on different time steps when generating the next sequence. I discuss how attention can alleviate the problem of long-term structure in music in its corresponding section in methods. I also employ the bidirectional wrapper for the LSTM cells, which tackles the challenge of long-term structure by considering both time directions as the context.

The third, **creativity**, relies on the system generating new pieces of music that exhibit a high degree of originality. Indeed, this is also important because of the danger of plagiarism. Unfortunately, as stated by the authors of the paper, there does not seem to be a viable solution for neural networks in this end.

I propose to measure the level of originality and plagiarism my models exhibit by using the MelodyShape tool [14]. MelodyShape is based on computing a geometric representation of the time-pitch dimension, and then calculating the similarity between these representations [15]. This representation is octave-invariant and tempo-invariant. The pitch slope can be measured by the difference in curvature. The algorithm can then be used to compare melodic music similarity between two monophonic MIDI files. The result is in the range of 0 - 1. I employ this to measure the overall plagiarism in the generated

sequences (see chapter on Experimental design).

The paper discusses the model by [52], an extension of the Generative Adversarial Network architecture[53][6]. They have adapted GANs so that the Discriminator provides two signals to the Generator: one that is found in GANs as well, which specifies how well the generated sample imitates the training set; and a second one, the innovation proposed by [52], which tells the generator how easily the discriminator can categorize the sample. The idea is thus that the harder it is to classify a sample the more original it is. In this sense, creativity is defined as "exploring new styles (which indeed has some grounding in the art history)" [23, p. 12].

The fourth challenge is **interactivity**. This problem arises out of the different ways in which musicians and neural networks work: musicians work in an iterative manner, with gradual "refinement and adaptation of a composition" [23, p. 13]; on the other hand, neural networks do not allow for regeneration and modification of previously generated samples. One solution to this problem is offered by the DeepBach system [10]. They propose a complex system combining LSTMs and traditional feed-forward networks. The advantage of this approach is that a user can re-generate a given sequence after changing a note in the surrounding time steps, in order to generate 4-voices chorales in the style of Bach.

I do not address this challenge in the present work.

Plagiarism detection in symbolic music

In this section, I will cover some of the aspects related to plagiarism and melodic similarity detection. This is a relevant question to answer. As the models for generating music are growing increasingly complex and capable we are in need of legislation and methods to handle the problem of originality, plagiarism, and authorship in this new context.[7]

In [54] the authors overview the main approaches to symbolic melodic similarity. These are divided into three larger categories: string-based methods, geometry-based methods, and rhythm and beat analysis. The first consists of efforts to treat music as strings, and use methods like edit distance measurements, substring matching, longest common subsequence to calculate similarity. In this case, we have Musipedia (http://www.musipedia.org/), an online database that relies on string edit distance and Earth Mover distance [55] [56].

The second method is grounded in seeing notes as points in a geometric space, with onset time, pitch, and duration, and then applying similarity measures to this space. This is the category to which MelodyShape belongs, one of the algorithms that has had the best results in the 2010 edition of the MIREX competition.[8] The algorithm relies on a geometric approach, embedding the melody into a pitch-time plane, being octave-, scale-, and tempo-invariant [15, p. 344]. An approach like this represents melodies as a quantifiable geometric object, where similarity can be computed using distance measures. Melodies are represented as sequences of n-grams. In fig. 2.4 we can see an example of such a geometric representation of the pitch-time continuum, based on the image from the original paper. In it, we observe an interpolating line passing through the points in the score. Thus, the similarity between two melodies is the similarity between the lines. The algorithm only works with monophonic MIDI files.

[6] Generative Adversarial Networks (GANs) are an architecture developed by Goodfellow [53]. It is formed out of a generator and a discriminator. The generator attempts to build fake samples out of a training set and the discriminator is trained to detect these fakes. The model grows as the two systems attempt to outperform each other.

[7] The topic is becoming an integral part of mainstream music culture. Popular YouTube vlogger and musician Adam Neely addresses the topic of AI-generated music in some of his videos (https://www.youtube.com/watch?v=vBvgHNPSYjQ, https://www.youtube.com/watch?v=xDqx14lZ_ls).

[8] https://www.music-ir.org/mirex/wiki/MIREX_HOME

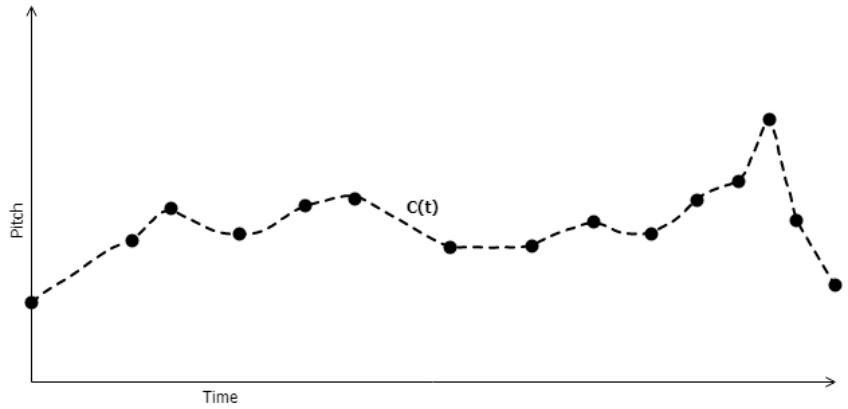


Figure 2.4: Representation of the pitch-time contour[9]

The interpolating line is referred to as C_t. In comparing two melodies we simply need to compare these two lines. The variation in the pitch dimension is measured by measuring the curvature difference. On the other hand, the variation in the time dimension can be solved analytically, as it relies on a linear transformation [15, p. 345].

[9]Image based on figure from [15].

3 | Methods & Architecture

In this chapter, I will discuss the different choice of algorithms, techniques, and learning models that I employ in my project. I justify their theoretical match to my problem and highlight ways in which they can structure the symbolic music composition task as a learning task.

I first provide an overview of the different deep learning techniques that I have employed in my model. Then I discuss the methodology of the experiments themselves. I discuss this more thoroughly in the chapter on Experiments.

Sequential model

I choose to employ a sequential Recurrent Neural Network (RNN) model. The model matches the musical structure logic. Ideas that develop in previous bars of music come back at later points, sometimes altered in different small ways. There are a lot of subtleties to the art of music composition. A sequential learning approach is one that can aim to capture this.

We can also think of music as a language. In this case, we can think of the multitude of sequential models that have been applied to natural language processing (NLP) tasks: sentiment analysis, text summarization, language modelling, text generation [57]. In a similar manner, we hope to capture some of the inner grammar of symbolic music with sequential LSTM models.

According to Douglas Eck, "in the case of music, long-term dependencies are at the heart of what defines a particular style, with events spanning several notes or even many bars contributing to the formation of metrical and phrasal structure" [58]. Thus maintaining an internal state in the RNN is required in order to preserve the global logic of a musical piece in a specific style. This is similar to a phrase in English, with the various relations between the parts can span multiple time steps (multiple words). In order to model these complex dependencies we need a sequential model: we need to consider previous melodic steps when composing the new one.

On the contrary, a simple "feed-forward network would have no chance of composing music in this fashion" [37, p. 2]. This is because such a network does not consider the time domain, and thus would not be able "to keep track of where it is in a song" [37, p. 2]. By modelling music as a sequence of symbols, similar to NLP language modelling tasks, we can apply RNNs.

Also, compared with a simple Markovian approach, deep learning allows the system to learn any degree of connections between the patterns. On the other hand, a Markov table must be built by the programmer with a certain n degree of connections. Such tables would grow exponentially as the user demanded more complexity from the model. Deep learning approaches eschew this problem altogether, allowing the system to adapt and learn potentially very elaborate relations between the data points.

Finally, as discussed in the literature overview on symbolic music generation, RNNs are the most commonly employed methods for the task.

LSTMs

However, RNNs do not perform well when the number of time steps is large. As discussed in the vanishing/exploding gradient section the gradient of the learning function tends to decay exponentially. This decreases the learning signal of the network, thus blocking any potential adaptations. There is also the possibility of an exploding gradient, in which the gradient grows exponentially. In order to alleviate these limitations, I choose to use the Long Short-Term Memory (LSTM) cell variant of the RNN.

LSTMs offer a solution to the vanishing gradient problem in the form of gates that control the flow of the learning signal. These control the input, output, and the internal cell state ("input" gate, "output" gate, "forget" gate, respectively). The LSTM learns to adapt to the flow of information in the data sequences. Also, by maintaining an internal state guarded by the "forget" gate it can be impervious to error noise in the data and still learn the essential patterns.

LSTM require longer training times and more computing resources, as there are more parameters to train to control the respective gates. However, LSTMs seem to have a major positive impact on sequential learning, when compared to traditional RNNs [43].

Gradient descent optimization

In order to optimize the learning process, I will apply different techniques, as described in the state of the art section on deep learning, in the appendix. These allow for faster, steadier, and better convergence of the model.

Firstly, I employ a **Dropout** layer after every LSTM layer. As mentioned, this allows the model to better fit the data by randomly turning cells on and off throughout the training process. This avoids overfitting. As discussed in [59] LSTMs and recurrent models in general exhibit a tendency to overfit the data. Thus, it is common to employ this technique for LSTMs, as used in [18], [60], [45]. From experimenting with different values, I choose a dropout rate of 40%.

While gradient descent is the basic version of the learning process in deep learning, there have been many proposed improvements to it throughout the years. I have chosen to use **Adam** (Adaptive Moment Estimation) [61], an adaptation of the RMSprop learner defined by Tijmen Tieleman in [62]. This optimizes the learning by "dividing the learning rate for a weight by a running average of the magnitudes of recent gradients for that weight". This greatly enhances the training process by keeping track of how steep a gradient each of the weights in the model has. RMSprop is also employed by [45], [63], [41]. Adam further builds on the advancements of RMSprop by combining it with the techniques of the Momentum learner [64, p. 4]. This latter method stores the gradually decreasing averages of the past slopes [64], allowing the learner to avoid overshooting the descending optimization path. As highlighted by Karphathy, Adam is one of the most popular gradient descent variations [65].

Finally, I employ a method of reducing the learning rate when the training stagnates for a given number of epochs. Reaching a plateau usually means that the function is slowly converging to a minimum (be it global or local), and that it is overshooting. By decreasing the step size (learning rate) we allow the function to descend into this minimum.

Bidirectionality

The bidirectional wrapper reflects the process of composing music as well, in the fact that a composer will plan in advance how the piece will end [66]. These plans are usually centred around the sense of home and resolved tension related to the tonic key of a piece. For example, a piece written in C Major will sound resolved when the piece returns to the note of C after having progressed through other notes of the scale. Thus a musician will sometimes work backwards from that during the process, in order to plan their moves across the scale and towards the final step. In music theory this is reflected by cadences [3]. These are progressions of chords with a specific movement between tension and resolution. Usually, the sense of resolution is at the end of the sequence. The composer will know of these theoretical concepts when writing their music. As such, they will plan their piece knowing where it will conclude. This sort of bidirectionality can be captured by a bidirectional LSTM network.

Bidirectional LSTMs are employed in [60], where the author uses them to embed the sequences from both directions. It is also used by Olof Mogren in [67]. In his system, he uses a bidirectional RNN discriminator in a GAN system. Thus the discriminator considers both time directions contexts for its decisions.

Attention Mechanism

The attention mechanism has been shown to improve sequential task performance significantly, as discussed in its respective section in the literature overview [18] [19] [20] [40]. It allows for the model to learn to focus on different steps of the temporal sequence when predicting the next given step. This is learned through the backpropagation process, like any other layer.

As this has been used with great success in other sequential tasks, including Natural Language Processing tasks, I propose that it will provide a boost to the performance of the model when generating new pieces. Music is by its very nature a sequential experience: you listen to each note one at a time (or several at a time in the case of polyphonic music) and within the context of what you have just had heard, in order to form a coherent whole. We can think of the call-response phrasing typical for blues music, where one initial phrase (the "call") is followed by the same phrase but with a slight change in the ending (the "response") [68]. A pattern like this can be difficult to learn by any model. By adding an attention mechanism to the learner, I propose an approach to storing and memorizing such a musical structure.

In symbolic music generation, it has been used by the Google Magenta team in their MelodyRNN model [17], with promising results. However, as pointed out, they did not provide enough detail on the implementation and experiments, as it was only a blog post.

Architecture

Based on the previous arguments I will propose, build, and analyze (objectively and qualitatively) several pipelines of datasets, encoding algorithm, and learning system architectures.

Thus, the work can be seen as a series of experiments where I change an independent variable (e.g. dataset encoding) and observe the dependent variable (the "quality" of the musical pieces produced by the model, as described by the quantitative analysis, plagiarism scores, and subjective discussion). There will be multiple such experiments, for the techniques proposed and discussed in the project. I provide an in-depth description of each of the experiments and how I define "quality" in this context in the following chapter, Experimental Design.

Sampling procedure

In order to generate new music I first choose a seed musical sequence. This needs to be a entirely new sequence, outside of the training and validation set. I then cut it to the first n time steps, n being the number of time steps in the input sequence. This is :

$$4(\text{bars}) \cdot 4(\text{beats per bar}) \cdot 4(\text{steps per beat}) - 1 = 63 \tag{3.1}$$

We deduct the last 1 step, as it is the note we predict.

After feeding the network this seed we will output 1 time step of musical information. This is then appended to the original sequence. We then move the input window by 1 time step to take the next 63 time steps as input for the next iteration. We continue this method until we obtain a certain number of time steps of original music. I choose this to be 4(bars) = 64(time steps). The choice of four bars is subjective, but rooted in musical composition theory. It is long enough so that a simple melodic theme can emerge.[1] Note that I do not keep the first generated time step (the one that is part of the four bars of the training sequence). I consider it to be part of the 4 bars used for training, forming a unit. I choose to generate 4 entirely new bars of music. Thus, I actually generate 65 time steps and discard the first one.

When sampling the next note we actually sample from a probability distribution p over the entire vocabulary. One option would be to simply select the token with the maximum probability argmax(p). However, this would limit the musical range of the network. Instead, we apply a temperature tweak to the softmax function, as discussed by Karpathy in [32]. This redefines the softmax function as :

$$\text{softmax}(x)_k = \frac{\exp(x_k)/\tau}{\sum_{i=0}^{K-1} \exp(x_i)/\tau}$$

where τ is the temperature. This has to be > 0. A temperature very close to 0 will simply choose the note with the highest probability. Increasing the temperature increases the randomness of the sampling procedure. Temperature can be adjusted in order to alleviate a model that has overfitted and has very high probabilities for a given next note.

Categorization

The problem we are solving is a supervised, sequential learning problem, with a many-to-one model (as per Karpathy's classification discussed in RNN types). We are feeding the model a sequence of musical time steps (63, as defined in eq. 3.1) and training it to predict the next step in the sequence. The arrows in the input sequence symbolize the sequential nature of the input. We can see in figure fig. 3.1 a graphical representation of the task. In light blue at the top, we have the input sequence. This is fed one step at a time into the LSTM (in light green in the middle). The model maintains an internal state that is passed from one time step to the other (symbolized by the dotted arrows in the model section pointing to the right). Finally, the model outputs a new time step of music (in dark blue, at the bottom).

[1] Several online instructional videos for making music deal with four bar loops as the basic unit in composing a song (https://www.youtube.com/watch?v=hG3ams7YtLU, https://www.youtube.com/watch?v=MN5GERlHcOM, https://www.youtube.com/watch?v=xk6nJPVkVB8)

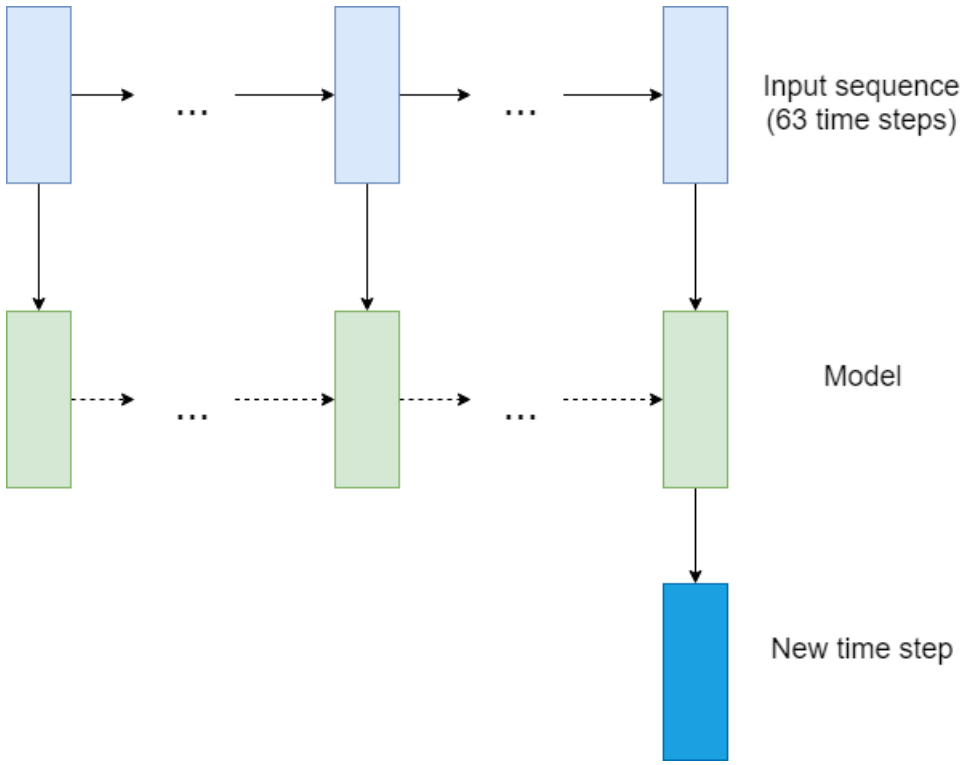


Figure 3.1: Music generation architecture

Analysis methodology

Quantitative analysis methodology

For each of the experimental comparisons, I will conduct a quantitative assessment of 100 musical pieces generated by the model, as compared with the training set. I follow the procedures described by Yang et. al in [13], which focus specifically on quantitative evaluation of symbolic music generation systems. They propose a framework for formative assessment of musical generation systems, by "using domain knowledge for designing human-interpretable evaluation metrics for generative music systems" [13, p. 4]. The approach is thus rooted in specialized knowledge of music theory and can also be easily understood by a human.

I will first extract 9 different musical features from the pieces. These are both pitch and rhythm-related, and have various dimensions. These metrics are then used for both absolute and relative measurements.

The nine features are as follows, with the acronyms that I will use in the tables. Yang et. al split them into two categories, pitch-related and rhythm-related. I preserve this categorization.

- pitch-related features:
  - PC: total used pitch/**p**itch **c**ount. The total number of different pitches in a sample. Represented by a scalar.
  - PR: **p**itch **r**ange. The difference between the highest and lowest pitches used within a sample. Represented by a scalar.
  - PI: average **p**itch **i**nterval. "Average value of the interval between two consecutive pitches in semi-tones" [13, p. 5]. A single scalar.
  - PCH: **p**itch **c**lass **h**istogram. Octave-invariant distribution of the pitch classes (notes in the chromatic scale) used in a sample. A vector of 12 values.

  - PCTM: **p**itch **c**lass **t**ransition **m**atrix. Distribution of pitch class progressions, i.e. how often does a note move to another specific note. A 2d matrix of 12 by 12.
- rhythm-related features:
  - NC: number of used notes (**n**ote **c**ount), independent of pitch. A single scalar.
  - IOI: average **i**nter-**o**nset **i**nterval. Mean time, in seconds, between two consecutive notes. A single scalar.
  - NLH: **n**ote **l**ength **h**istogram. Similar to PCH, but with note lengths. The rhythmic subdivisions are: whole note, half, quarter, eighth, sixteenth, dotted half, dot quarter, dotted eighth, dotted sixteenth, half note triplet, quarter note triplet, eighth note triplet. We get a vector of 12 numbers.
  - NLTM: **n**ote **l**ength **t**ransition **m**atrix. Similar to PCTM, but with the above 12 rhythmic subdivisions. We get a 2d matrix of 12 by 12.

It is important to note that with regards to the rhythmic features, I am limiting the dataset to only those pieces in 4/4 time signature[2]. I am also ignoring issues related to tempo differences, as I do not encode that in any way in the dataset formats.

The absolute measurements are based on calculating the mean and standard deviation of these features. They then yield relevant information about the training dataset qualities and general characteristics [13]. However, these values each have different scales and must be interpreted differently. Thus, for relative measurements, Yang et. al suggest using Euclidean distances.

The relative measurement is based on computing the Euclidean distance between each sample and all the other samples in its own dataset (*intra-dataset*). This results in histograms for each feature. I then compute a probability distribution function of each feature "in order to smooth the histogram for a more generalizable representation" [13, p. 6]. Then, the mean value of the *intra-set* distances can be interpreted as the average variety in those samples of that feature. The mean value of the *inter-set* distances can be read as the average similarity between the two datasets, on that feature. Standard deviation can then be understood as the "reliability of mean value" [13, p. 6].

To compute the similarity between two datasets I compute the similarity between the distributions. I use Overlap Area (OA) and Kullback-Leibler Divergence (KLD) between the distributions of the *intra-dataset* distances.

KLD is defined as follows:

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log \left(\frac{P(x)}{Q(x)} \right)$$

Where P and Q are the distributions. In my case, this is the distribution of distances within a certain dataset of a given feature.

Thus KLD would be 0 between two identical distribution, while OA would be 1. The authors mention the problematic nature of KLD, since it is "unbounded and asymmetric" (p.6). So $D_{KL}(A||B) \neq D_{KL}(B||A)$. I then employ OA as a "bounded measure in the range $\in [0, 1]$" (p.6).

OA can be calculated as the integral of the minimum between the two distribution functions:

$$OA = \int \min[f_1(x), f_2(x)] \mathrm{d}x$$

[2]For a complete overview of the preprocessing steps, see the section Preprocessing in the Dataset chapter.

where f_1 and f_2 are our two distributions.

In interpreting the results I focus on the differences between the datasets, but also in the *differences between the differences*. Taking the case of comparing two models, I first compare how similar a model's samples are to the training set, then how similar the other model's are to its own dataset. Finally, I compare which of the models is closer to its training set, in terms of inter-set distances (using OA and KLD).

I also compute the overall inter-set similarity per each *category of features* (pitch and rhythm-related), and an *overall similarity* across all features, in terms of KLD and OA.

Plagiarism detection

However, one can expect that an LSTM model that achieves a high degree of similarity with the training set might just be repeating melodies from that set, without exhibiting originality. This can indeed be a problem, as discussed in the overview of the four challenges in the field of music generation (pg. 14). I provide an objective measure of this. I employ the MelodyShape algorithm [14] [15] to measure the melodic similarity between the generated set and the training set.

For each sample that I generate with a model, I traverse the training set and compute the similarity score with the algorithm. The algorithm outputs a similarity score in the scale 0 - 1. I then store the highest score for each generated piece. Thus, for a set of 100 generated samples, I will obtain a list of 100 scores in the range 0 - 1. I then take the average of this to represent the overall plagiarism score for a given model. I also provide the standard deviation. I discuss the algorithm in the section on Plagiarism detection in symbolic music, and I provide a further description of the process in the Experimental design.

User study methodology

I also perform a user study assessment of the samples produced by one of the models I develop, as compared with samples from the training set. For this, I develop a user survey and ask people to rate the music pieces. I discuss this in more detail in the section on User evaluation

Subjective evaluation

For each of the experiments, I also provide several samples of generated music in sheet format. I then discuss the features and style of these pieces, from a subjective perspective. To qualify my experience, I have been playing the guitar for about 10 years, but never at a professional level. I have also been learning music theory and composing electronic music for about 3 years, again at an amateur hobbyist level. I am not an expert in music theory or composition, but I believe I can provide some insight into the pieces.

For all of these pieces, I use as a seed sequence fours bars of music composed by myself. I do this in order to experiment with one of my initial motivations for this project. As already stated in the motivation section in the introduction, I am interested in exploring machine learning models for music generation as a tool to enhance the musician's creative process. The goal here is not to completely replace the human aspect. In this case, I place myself in the seat of a composer wanting to generate new ideas in order to continue one of their pieces.

The composed piece I use as a seed sequence can be seen in figure fig. 3.2. It is a relatively simplistic piece, based in the key of A minor. It's based on a repeating arpeggio pattern for the first three bars, with a resolution to the root key in the fourth bar.

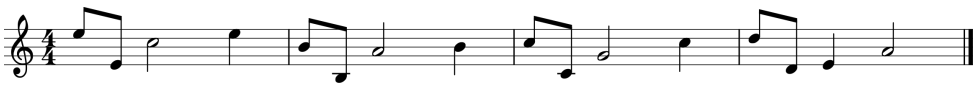


Figure 3.2: Original composition for seed sequence

4 | Dataset

In this chapter, I will discuss the datasets that I have used for training the models. I will provide an overview of their style, their origin, and conduct an objective measure comparison using the approach from [13]. This is done in order to highlight the differences and similarities between the datasets.

Datasets are an essential part of my problem statement. In the experiments section, I analyze the effect of music dataset style homogeneity on the output of the generative model. In my case, I compare two datasets: one that is stylistically homogenous (traditional folk tunes) against one that is stylistically heterogeneous (a combination of rock, pop, jazz, classical).

Like all deep learning problems, the dataset is crucial in obtaining a good model. In symbolic music generation, this is made more difficult by several problems. Firstly, there are very few quality datasets. By quality dataset, I am referring to a dataset with a significant amount of examples, but with also good meta-information about the tracks: instruments per track, genre tags, chord progression etc. For the purposes of this project, I have chosen two datasets where I could easily extract the melodies.

A second problem is the lack of a standardized way of preprocessing and encoding the dataset. There are multiple processes and formats proposed, each with their own advantages and disadvantages. However, the research in the field rarely provides adequate detail on how the dataset was processed. Even fewer projects release the dataset or code for the preprocessing of the dataset. This makes the pursuit of a standard in the field more difficult. For the purposes of this project, I have chosen to experiment with two dataset encoding formats. I have also outlined in great detail the preprocessing procedure for my data.

Description

Folk dataset

Firstly, I use a dataset of traditional Irish folk tunes. This was scraped from the website https://thesession.org/. This contains a collection of different types of songs, jigs, reels, polkas, waltzes etc., contributed by a community of users. They are originally in ABC notation format [44]. However, I have used the MIDI version made available by Ira Kurshunova on GitHub.[1] This contains 46000 MIDI files, of varying lengths and time signatures.

I consider this dataset to be stylistically homogenous. They all belong to one single genre of music, all of them being traditional folk music. This is contrasting to the second dataset, which is an amalgam of various genres.

[1] https://github.com/IraKorshunova/folk-rnn

Hook dataset

The second dataset is based on the lead sheets from https://www.hooktheory.com/. The site hosts approximately 12000 lead sheets.[2] These are usually chord progression and main melody, and are divided by songs sections. The songs themselves are across multiple genres: from classical to metal to jazz. This makes it a very heterogeneous dataset.

I have scraped this from the website using an open source toolbox from GitHub.[3]

Statistics and dataset comparison

In this section, I will present important statistical highlights from the datasets. These will help the reader better understand how each dataset is unique, and how, in certain ways, they are similar. These are domain-specific statistics, inspired by musicology and music theory. These have been implemented in different libraries and specialized software, including *JSymbolic* [69] and the Python toolbox provided by [13], to which I will refer to as *mgeval*. In this current work, I am employing the methodology explained in the [13] paper. The authors provide a thorough explanation of different approaches to quantitative analysis of music generation systems. I discuss this in more detail in the respective section on quantitative methodology on page 21.

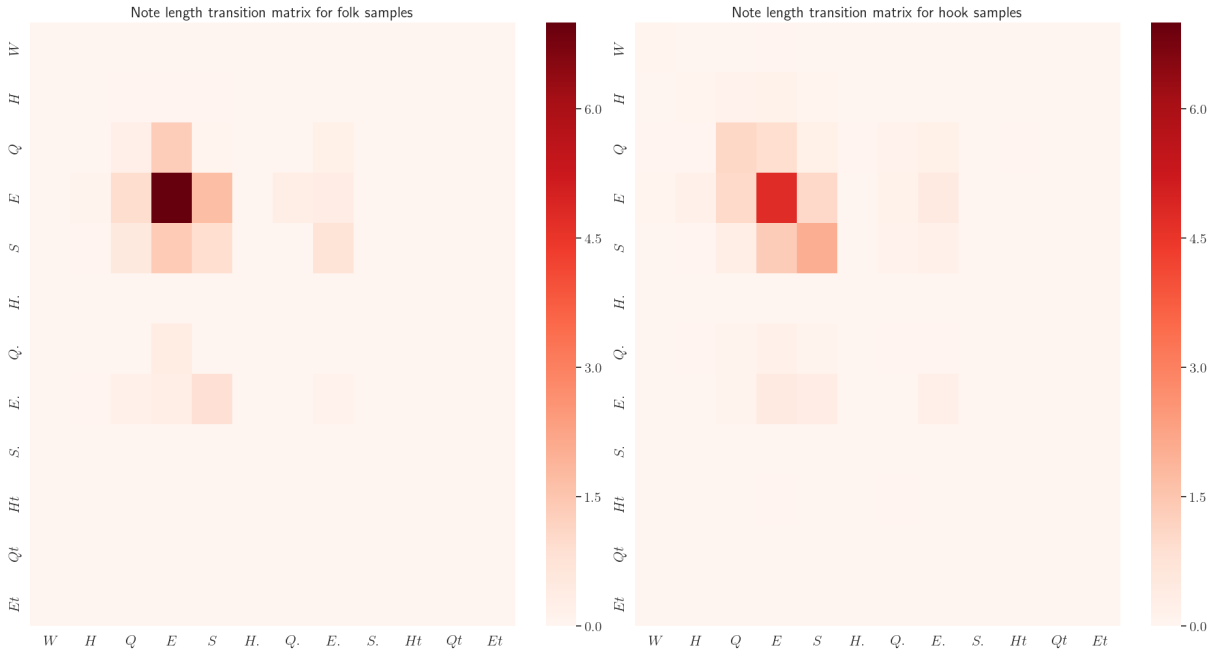
Table 4.1: Comparison of Folk and Hook datasets

	Folk				Hook				Inter-set	
feat.	**mean**	**SD**	**intra-set mean**	**intra-set SD**	**mean**	**SD**	**intra-set mean**	**intra-set SD**	**KLD**	**OA**
PC	8.390	1.849	2.087	1.597	5.840	2.444	2.408	2.504	0.632	0.762
PR	14.110	3.493	3.935	3.026	10.110	4.872	4.908	4.885	0.116	0.833
PI	2.796	0.647	0.724	0.568	2.687	1.499	1.410	1.596	0.258	0.736
PCH	-	-	0.316	0.110	-	-	0.485	0.163	0.035	0.538
PCTM	-	-	8.081	1.645	-	-	6.191	2.289	0.488	0.548
NC	28.070	3.567	3.976	3.147	17.990	8.311	8.651	8.043	0.174	0.620
IOI	0.286	0.042	0.043	0.042	0.523	0.379	0.305	0.443	0.201	0.322
NLH	-	-	0.354	0.264	-	-	0.597	0.263	0.456	0.540
NLTM	-	-	12.092	6.494	-	-	11.489	8.685	3.597	0.869

See table 4.1 for statistical comparison of the two datasets. In this, we can see that indeed the folk dataset is more stylistically consistent. This is due to the standard deviation of the features in the folk dataset being lower than in the case of the hook dataset. This is the case for both the absolute measurements (*mean* and *SD*) but also for the intra-set distances (*intra-set mean* and *intra-set SD*). As pointed out by Yang et. al, standard deviation "serves as an indication of the reliability of mean value" [13, p. 6].

Another interesting observation can be made about the features with the highest and lowest overlap area. Firstly, the feature with the lowest OA is the IOI, average inter-onset seconds between two notes. We observe that the SD for this feature is much smaller in the case of the folk dataset. This might represent that the folk dataset is more consistent in its rhythmic patterns. Indeed, if we look at the note length transition matrices in fig. 4.1, we notice that the hook dataset is characterized by a more diverse rhythmic content. The folk dataset seems to be characterized by a focus on eighth notes, while the hook dataset is more evenly spread. This, along with the previous observation about the overall lower standard deviations, further confirms that the folk dataset is more stylistically consistent.

[2] As of 2019-02-10.
[3] https://github.com/wayne391/Lead-Sheet-Dataset



(a) NLTM for folk dataset

(b) NLTM for hook dataset

Figure 4.1: NLTM for datasets. The labels represent, in order: Whole note, Half note, Quarter note, Eighth Note, Sixteenth Note, Dotted Half, Dotted Quarter, Dotted Eighth, Dotted Sixteenth, Triplet Half, Triplet Quarter, Triplet Eighth

This can be further observed in fig. 4.2. The intra-folk distribution of distances on the IOI feature is all very clustered around the mean, symbolized by the very low SD. On the other hand, the intra-set distances for the hook set are more spread out.

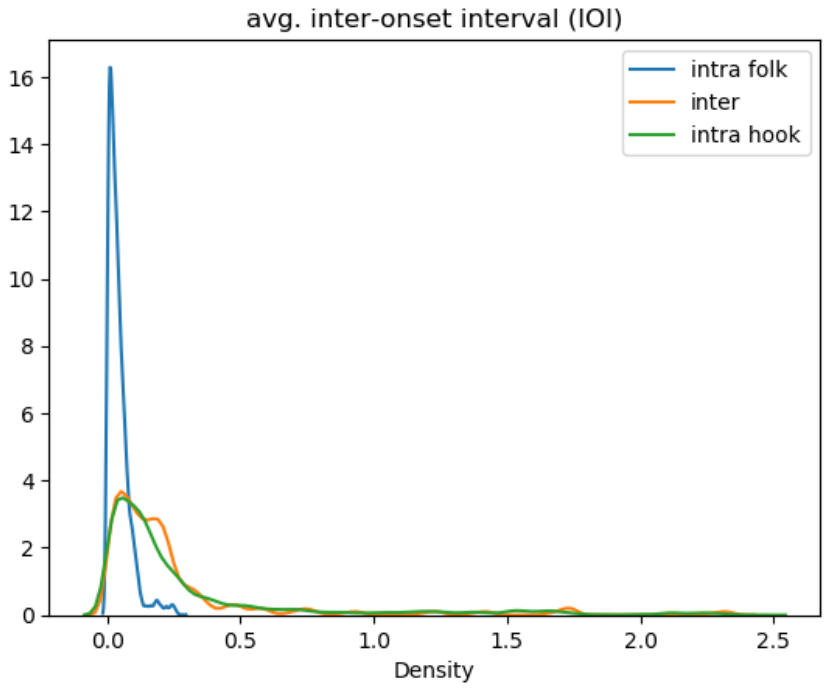


Figure 4.2: Distribution of distances on IOI feature

Secondly, the features with the highest OA are Note Length Transition Matrix (NLTM) and Pitch Range (PR). The fact that the two datasets have a high OA for the NLTM feature seems to contradict the differences in the fig. 4.1. However, if we look at the Kullbeck-Leibler divergence (KLD) we can see that it is very high for NLTM and very low for PR. From this, we can deduce that, in fact, the datasets are indeed different with regards to rhythmic content (as represented by NLTM and IOI).

The high OA for PR, backed by a low KLD, means that the datasets tend to use a similar range of notes across each sample. However, if we use at PI (average pitch interval between consecutive notes) and PC (number of distinctive octave-independent pitches used in a sample) we can see that the hook dataset exhibits a higher SD (as already discussed). This can be interpreted as, even though we have a similar PR, the most commonly used notes are actually quite different between the datasets.

We can conclude from this discussion that overall the folk dataset is more consistent. This is because it consists of musical pieces from the same genre and styles.

Formats

In this section, I will discuss one of the challenges faced by research in symbolic music generation: dataset encoding. I will present the need for such an encoding and then compare several options.

While music is encoded as audio files (`.mp3`, `.wav`) neural networks require numbers as input. In order to achieve this, we are required to encode the data into a digital format. The first step is transcribing the music into MIDI, which is the equivalent of sheet music but in digital format, as discussed in its own section. This is done by hand by the user, using different software.[4]

The next step involves taking the MIDI files, which are in a specialized encoding that is not understandable to neural networks, and further encoding them into a matrix shape that allows linear algebra operations to be performed on it. Unfortunately, unlike other machine learning tasks like NLP and image processing, there is no de-facto standard on symbolic music encoding for deep learning.

Some of the common formats are pianoroll, magenta *melody* (from [17]), Transformer score (from [70]). For the purposes of this project, I will restrict myself to discussing and experimenting with the former two.

Pianoroll format

The closest there is to a standard is the **pianoroll** format. This aims to represent music notes as the rows in a matrix, with the time steps being the columns. When a note is being played, the value of $M_{p,t}$ is 1 (*on*). Otherwise, it's 0 (*off*). In this case M is a 2d matrix with $p = 128$ number of pitches and t number of time steps. This is modelled after the representation of MIDI notes in DAW software, as seen in fig. 6.8.

The time steps are calculated based on a quantization measure, measuring how many time steps should each beat be represented as (a fourth of a bar in a 4/4 time signature). This choice dictates the granularity with which the rhythmic variations of a piece might be represented. I choose $q = 4$, with which I can capture 16th notes. This allows for a high level of detail.

It's close to being a standard because it is so widely used. Pianoroll representation is the most popular representation, being used in a wide range of papers: [71] [72] [73] [74] [75], to cite a few. The simplicity

[4] A free open-source option is MuseScore (https://musescore.org/).

of the format arises out of its correspondence to DAWs representation and the existence of libraries that parse and manipulate these.[5]

Note ending problem

However, the pianoroll format has one limitation with regards to its encoding. Two notes of length l of the same pitch being played in consecutive order are indistinguishable from one note of length $2l$. So eight eighth notes of the same pitch are represented in the same way as four quarter notes of the same pitch. While in the MIDI file the ending is signalled by a NOTE OFF event, this is not preserved in the pianoroll encoding.

This problem is also discussed in [11, p. 24], where the author mentions two solutions:

The first one is to divide the size of the time step quantization by 2 and always replace the last step in a note's representation with a 0 instead of the usual 1 [37]. This is by far the most common solution. The advantage is that this does not require any other tags. However, there is a problem with this approach: when re-encoding the matrix back into MIDI format the rests are preserved, even though they should not be interpreted as rests. Thus, a sixteenth note will be interpreted as 3/4 of a sixteenth note, with a 1/4 of a sixteenth note rest at the end. This creates a more fragmented musical piece, which is perhaps not desirable.

The second is to have another special tag in the dataset that symbolizes the MIDI NOTE ON event. This is first suggested by Todd in [78]. This is what the *melody* format is based upon.

Melody format

Another encoding format is proposed by Google Magenta in their MelodyRNN models [17]. I will refer to this format as the *melody* format, for convenience. This is similar in pianoroll with regards to its representation of the time dimension. A musical piece is still represented as a sequence of vectors, with beat quantization controlling how many steps a quarter note receives.

The format works similarly to the pianoroll, by encoding each pitch as its own row in a matrix M of p pitches and t timesteps. The difference between the *melody* and pianoroll format is that the *melody* format does not hold the 1 flag (*on*) for the entire duration of the note. Rather, it has another token that symbolizes that no new musical event is happening. This means that the last note is being held. When a new note is triggered (a value of 1 in one of the p pitch rows), the last note will automatically be turned off (MIDI NOTE OFF event).

This format ensures monophony even when generating a new sequence, and that I will not need to do any extra postprocessing to extract the melody. It does so by always stopping whatever note is playing when a new NOTE ON event is issued. Most other encoding formats (including the Transformer score [79] and pianoroll [11, p. 19]) encountered do not enforce monophony through the rules themselves. Strict monophony is required by the melody plagiarism detection tool [14].

In the experiments section, I provide a comparison between a model trained on a dataset encoded with the pianoroll format and a model trained on the same dataset but encoded with the *melody* format.

[5] *pretty_midi* [76] and *pypianoroll* [77].

Preprocessing

In this section, I will discuss the preprocessing techniques I have applied to the dataset, in chronological order of how they are applied. These procedures are important for reproducibility, as even the smallest difference can cause differences in results. All the techniques are based on domain knowledge and are music-specific. It is in general important to have a very deep understanding of the field your data belongs to before you use it.

Firstly, I set the velocity of all notes to the maximum 127, thus abstracting away the need to learn velocity and note dynamics expression. Thus, each note is always played at the same velocity.

I choose to filter out the datasets to only the pieces composed in 4/4 time signature. This means that each bar of music is divided into 4 quarters, out of which we play 4. This is the most common time signature in popular music genres (classical, jazz, pop, rock etc.). I choose to do this so that the system only focuses on learning one system of divisions and subdivisions. Indeed, depending on the time signature, each temporal step could be interpreted as either the beginning, middle, or end of a bar. In certain genres, each step in the bar (divided into upbeat and downbeat) have certain conventions about what sort of harmonic or melodic content they should contain (See [1]). Thus, filtering to 4/4 simplifies the learning process understanding to just one time signature and its associated conventions.

I transpose all the songs to the keys of C Major, or its relative minor, A Minor. I do this again in order to simplify the learning process for the model. By transposing all the melodies to one key the model only has to learn the rules of melody composition in that one key. The compositions can then be easily transposed to all the other keys. While it is indeed important to consider the key when composing a musical piece, as each key evokes a different mood, for the purposes of this project I will restrict the model to only learning the rules of melody writing itself, abstracted from the key.

I extract monophonic melodies from each of the tracks. Monophony means at any given point there is only one note being played. This is crucial in order to ensure that the model is exclusively focused on the part of melody writing, and more specifically on monophonic melody writing. If I do not ensure that all MIDI files are monophonic I also cannot employ the MelodyShape algorithm for musical similarity detection, which requires strictly monophonic melodies. I extract the melodies by progressing through the notes in the track and terminating the ongoing note whenever a new note is being triggered. For the hook dataset, I also make sure to remove the chord track and only use the melody track. The pieces in the folk dataset, on the other hand, are only melodies.

I choose to optimize the data memory usage by limiting the encoder to the minimum and maximum notes encountered in the dataset, thus speeding the learning process, as the model does not need to learn about the unused pitches. In the folk dataset, this is quite sizable, decreasing from 127 (the standard MIDI range) to just 58 used pitches. In the case of the hook dataset, it's 90. The difference in range can be attributed to the difference in stylistic diversity, as some genres tend to do use different pitch range (e.g. compare classical music played on cellos with pop music written for upper vocal ranges).

I choose to set the maximum sequence length to 4 bars. This allows the model to develop a sense of melody. I create multiple 4-bar sequences out of the files in the datasets by moving a 4-bar window across each file at a time, with a step size of 1 bar. This means, e.g. that a MIDI file that is 5 bars will have two 4-bar sequences.

I use a beat quantization of 4. This means that every beat (a fourth of a 4/4 bar) will be represented by 4 time steps in the data. Thus, the lowest quantization will be 16th notes. This does not allow for triplets.

For each of the two data sources (*folk* and *hook*) I create a dataset of 100000 4-bar long sequences, which I will then split into training and validation sets for model training. It is important to note that the model trained on the hook dataset will be attempting to learn m genres, as opposed to one single genre in the folk dataset, from the same amount of n samples. I will discuss the consequences of this in Experiment 1.

Data augmentation with multiple key transpositions

A common procedure for data augmentation in music is to transpose the dataset of melodies into all the keys. This is done in [71] [50] [51]. However, as discussed in [36], this does not result in aesthetically pleasing results within the same amount of training time. This is because the network now needs to learn patterns of musicality across multiple keys at the same time. I choose to restrict the melodies to one key (C major and its relative minor, A minor). In this way, the network only needs to learn melodic patterns within one key.

5 | Results & Discussions

In this chapter, I provide a thorough discussion and analysis of each of the experiments outlined in the previous section. I detail the purpose of each of these, provide tables and plots, and draw conclusions in the context of the task of symbolic music generation.

Experiments

In this section, I will describe the experiments that I have done in this project. In each of the experiments, I test for one independent variable (e.g. whether the system employs or does not employ the attention mechanism) and observe how that affects the dependent variable (the quality of the results, as defined by the quantitative and subjective analyses).

To restate my hypotheses, as outlined in problem statement:

1. the nature of the dataset can influence the results of a generative deep learning model.
    a. Does stylistic homogeneity affect learning? This is quantified by the diversity (in terms of standard deviation) across the features describing the dataset. Qualitatively, it is the genre coherence of the dataset. One dataset is composed of an amalgamation of genres (pop, rock, jazz, classical etc), while another is made up of songs belonging to traditional Irish folk music. The latter one is the homogeneous one. This hypothesis is tested in experiment 1.
    b. Does the encoding format used for representing the dataset influence learning? These are the rules for converting the music from MIDI format to a format that can be processed by the model, i.e. that can be used in linear algebra operations. I provide a comparison between the "traditional" pianoroll encoding and the *melody* encoding proposed by the Magenta team [17] in experiment 2.
2. different sequential deep learning techniques can improve the learning of the model
    a. Does the attention mechanism, as described in page 10, improve the quality of the results? This mechanism has been shown to help sequential models learn which sequence steps are most relevant in predicting the output. This is tested in experiment 3.
    b. Does bidirectionality improve the learning of the LSTM model? Bidirectionality works by making the model aware of the entire context of the sequence, both backwards and forwards in time. This is tackled in experiment 4.

The process was as follows for each experiment, except where otherwise noted. I first train the models using the architectures specified in each experiment. All the models are implemented in the *Keras* library [80]. They are all trained for 50 epochs, with an *Adam* optimizer with an initial learning rate of 0.005. I employ a mechanism of lowering the learning rate when the validation loss plateaus, with `factor=0.5, patience=3, epsilon=0.0005`. The first parameter symbolizes by which amount the learning rate changes, the second symbolizes how many epochs should the mechanism wait before

changing the l.r., and the last one symbolizes the difference by which the validation loss should change in order to be considered. I use a batch size of 128.

I then extract quantitative measures from 100 samples generated by each of the models, and provide a discussion based on these. The samples are generated with a temperature of 1.0 and based on a seed sequence of 4 bars from the song "Raining Blood" by the thrash metal band Slayer, transposed to the same key as the dataset (C Major / A minor). I choose this sample since it does not exist in either dataset and is outside of the training sets' stylistic range. It is thus an approximately equal challenge to any model trained on these datasets to generate a matching piece based on this. If I had chosen a piece from a genre within the datasets I would have biased the evaluation to benefit that system.

For extracting plagiarism scores I run the MelodyShape algorithm on the 100 samples of each model, against the training set of the respective model. I restrict it to the most similar result per sample. I thus get a list of 100 scores per model. I then compute the statistics for this, mean and standard deviation. In some cases, the algorithm runs into an error and outputs $-\infty$ as the similarity score. By examining the MIDI files in the generated batch, we observe that these files do not hold enough information in order to compute a meaningful similarity score. The MelodyShape algorithm works with 3-grams, so that requires a minimum of 3 notes. In some cases, a sample will not even have 3 notes. Thus I choose to ignore these and replace them with the mean of the other data points.

I have already outlined the subjective analysis on page 23.

I discuss each experiment and its results in the following.

Experiment 1: dataset heterogeneity

In this experiment, I train two models with the same architecture, but on different datasets: the folk and the hook datasets. I then evaluate how the choice between a homogenous (folk) and heterogeneous (hook) dataset affects the features of the samples generated by the respective models

The architecture is as follows:

- input layer. This layer differs in the number of vocabulary tokens, depending on the dataset. This is because I have chosen to trim the vocabulary of MIDI notes to just the ones actually used in the music. Thus, the folk dataset has a smaller vocabulary range (58) than the hook dataset (90). I use 63 time steps, which is, as discussed, 4 bars of music without the last timestep (see eq. 3.1 in Sampling procedure)
- 2 hidden layers of bidirectional LSTMs, with 64 cells each. The bidirectional wrappers double the number of cells by adding a negative time direction LSTM. Thus we have 128 cells on each layer
- each layer is followed by a dropout layer, with a 40% rate. As discussed in the section on Gradient descent optimization in the chapter on Methods, dropout allows us to mitigate overfitting.
- attention layer on top of the last LSTM layer
- output layer. This has the same vocabulary range as the input layer, but with only one time step, as opposed to 63.

Objective Analysis

We can see a comparison between the folk dataset and the model samples in table 5.1. We can observe that the model trained on the folk dataset is better at emulating the pitch-related features, rather than rhythmic ones. This is based on a higher OAs on the Pitch Count (PC), Pitch Class Histogram (PCH),

Table 5.1: Experiment 1: Training set (folk) and model comparison

	Training set (folk)				Model				Inter-set	
feat.	**mean**	**SD**	**intra-set mean**	**intra-set SD**	**mean**	**SD**	**intra-set mean**	**intra-set SD**	**KLD**	**OA**
PC	8.390	1.849	2.087	1.597	7.650	1.915	2.105	1.726	0.149	0.863
PR	14.110	3.493	3.935	3.026	13.230	4.226	4.538	3.935	0.086	0.750
PI	2.796	0.647	0.724	0.568	2.558	0.499	0.571	0.420	0.007	0.847
PCH	-	-	0.316	0.110	-	-	0.293	0.121	0.031	0.888
PCTM	-	-	8.081	1.645	-	-	9.145	2.619	0.048	0.761
pitch avg.	-	-	-	-	-	-	-	-	0.064	0.822
NC	28.070	3.567	3.976	3.147	30.790	1.314	1.435	1.194	0.466	0.438
IOI	0.286	0.042	0.043	0.042	0.260	0.012	0.013	0.011	0.401	0.459
NLH	-	-	0.354	0.264	-	-	0.059	0.043	0.801	0.205
NLTM	-	-	12.092	6.494	-	-	3.939	2.915	0.196	0.377
rhythm avg.	-	-	-	-	-	-	-	-	0.466	0.370
overall avg.	-	-	-	-	-	-	-	-	0.265	0.596

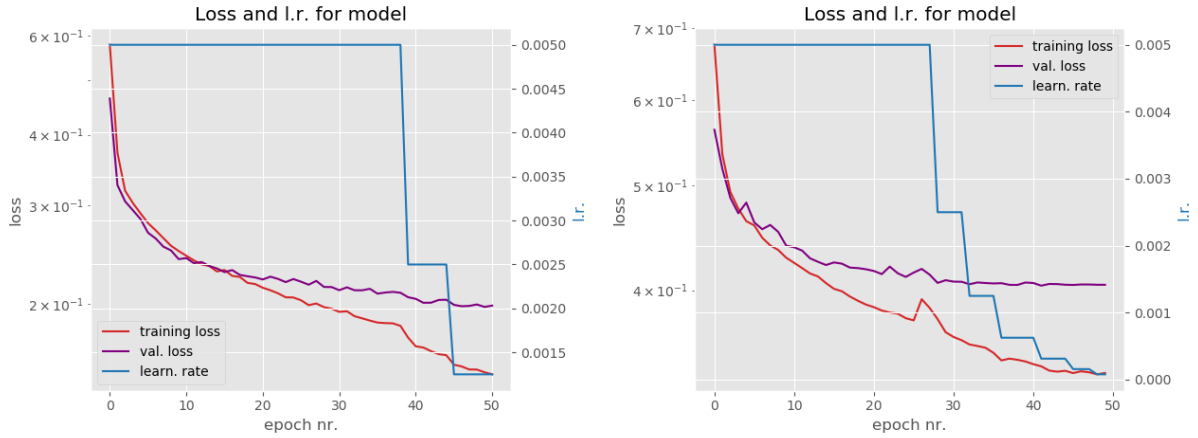
Table 5.2: Experiment 1: Training set (hook) and model comparison

	Training set (hook)				Model				Inter-set	
feat.	**mean**	**SD**	**intra-set mean**	**intra-set SD**	**mean**	**SD**	**intra-set mean**	**intra-set SD**	**KLD**	**OA**
PC	5.840	2.444	2.408	2.504	2.390	2.039	1.929	2.163	0.160	0.748
PR	10.110	4.872	4.908	4.885	3.370	5.081	4.717	5.468	0.043	0.804
PI	2.687	1.499	1.410	1.596	2.895	4.034	2.773	3.295	1.425	0.453
PCH	-	-	0.485	0.163	-	-	1.037	0.404	1.395	0.199
PCTM	-	-	6.191	2.289	-	-	2.102	2.437	1.683	0.191
pitch avg.	-	-	-	-	-	-	-	-	0.941	0.479
NC	17.990	8.311	8.651	8.043	3.910	7.064	4.546	8.953	0.385	0.508
IOI	0.523	0.379	0.305	0.443	0.821	1.022	0.944	0.705	0.891	0.294
NLH	-	-	0.597	0.263	-	-	0.801	0.464	0.682	0.506
NLTM	-	-	11.489	8.685	-	-	4.032	8.137	0.849	0.199
rhythm avg.	-	-	-	-	-	-	-	-	0.702	0.377
overall avg.	-	-	-	-	-	-	-	-	0.821	0.428

and Average Pitch Interval (PI) features, compared to features such as Note Length Histogram (NLH) and Note Length Transition Matrix (NLTM).

We can see a comparison between the hook dataset and the model samples in table 5.2. We can see that the model has a poor capability of emulating both the pitch and rhythmic aspects of the dataset, due to its overall low OA. Overall, from the quantitative evaluation, the folk model is better at emulating its training set than the hook is at emulating its training set, especially on the pitch features.

Comparing it to the training set's (hook) absolute statistics for Note Count (NC), we observe that the model produces samples with fewer notes (~4 compared to ~18). This signifies that the model is having problems producing new notes, and instead plays fewer, longer notes. Indeed, in the subjective analysis to follow, I observe this pattern. A reason for this might be the high variety of samples in the dataset, thus rendering the model unable to converge and learn a systematic vocabulary of musical phrases. By looking at the loss history in fig. 5.1 we notice that the hook model stagnates at a loss of ~0.4, while the folk model achieves a lower loss of ~0.2. This is expected, due to the higher diversity of songs in the hook dataset limiting the generalization of the model from the training set to the validation set.



(a) Experiment 1: Loss graph for folk model

(b) Experiment 1: Loss graph for hook model

Figure 5.1: Experiment 1: Loss graphs for models

We can see that the models tend to converge to a specific point on the validation set, but continue to learn from the training set. In the case of the hook model, this difference between the validation and training set is more pronounced, as compared to the folk model. This can be interpreted as overfitting[1]. Thus, the model might have developed confident weights assigned to specific note probabilities. To alleviate this, I generate another 100 samples with the hook model, but this time with a temperature of 1.5. As discussed in Sampling procedure, a higher temperature increases the entropy of the sampling process, yielding more random samples. In our case, it is used to avoid choosing the note with the highest probability. We can see the statistics for this model in 5.3. Indeed, we can notice that the model has now a much higher OA for both pitch feature and rhythm features, as compared to table 5.2. However, this also increases the instability of the model. This can be seen in the higher standard deviations for all the absolute measurements: SD for Pitch Range (PR) is ~10.5, compared to training set of ~5; SD for Note Count (NC) is ~12.6, compared to ~8.3 for training set. This renders the results hard to interpret. As highlighted by Yang et. al in [13], the standard deviation "value serves as an indication of the reliability of mean value" (p.6). It can be that raising the temperature has yielded some results that are indeed more "musical", but it has also yielded more random results in general.

Table 5.3: Experiment 1: Training set (hook) and model comparison (t=1.5)

	Training set (hook)				Model (temperature = 1.5)				Inter-set	
feat.	**mean**	**SD**	**intra-set mean**	**intra-set SD**	**mean**	**SD**	**intra-set mean**	**intra-set SD**	**KLD**	**OA**
PC	5.840	2.444	2.408	2.504	9.320	4.714	5.232	4.185	0.114	0.585
PR	10.110	4.872	4.908	4.885	18.500	10.570	11.769	9.338	0.069	0.562
PI	2.687	1.499	1.410	1.596	3.633	1.730	1.843	1.601	0.092	0.757
PCH	-	-	0.485	0.163	-	-	0.554	0.220	0.040	0.838
PCTM	-	-	6.191	2.289	-	-	7.028	3.103	0.169	0.856
pitch avg.	-	-	-	-	-	-	-	-	0.097	0.720
NC	17.990	8.311	8.651	8.043	20.060	12.600	14.171	10.950	0.212	0.721
IOI	0.523	0.379	0.305	0.443	0.501	0.668	0.437	0.831	0.303	0.671
NLH	-	-	0.597	0.263	-	-	0.568	0.288	0.020	0.919
NLTM	-	-	11.489	8.685	-	-	14.450	11.129	0.279	0.758
rhythm avg.	-	-	-	-	-	-	-	-	0.204	0.767
overall avg.	-	-	-	-	-	-	-	-	0.150	0.743

[1] One solution could also have been to employ an early stopping mechanism. For all my networks I have decided to instead use the mechanism for reducing the learning rate in order to force the model to converge to a better optimum.

Plagiarism and originality

In table 5.4 we can see the statistics for the plagiarism scores. We observe that the folk model exhibits the lowest score of plagiarism. This is further enforced by a low standard deviation as well. On the other hand, the model trained on the heterogenous hook dataset exhibits a higher plagiarism tendency. The column denoted by `-inf` represents the number of samples in that batch of 100 which did not contain enough musical information to have their similarity score computed by MelodyShape. The algorithm requires at least three notes. These have been replaced by the mean of the other data points. Since there are so many (more than half) samples that could not be interpreted by MelodyShape, it is difficult to ascertain how this model is performing with regards to originality.

Table 5.4: Experiment 1: plagiarism scores for models

model	mean	SD	-inf
hook(t=2.0)	0.422	0.335	0
hook(t=1.5)	0.550	0.351	5
hook(t=1.0)	0.812	0.154	62
folk(t=1.0)	0.138	0.064	0

In order to obtain a meaningful metric for plagiarism, I then also sampled two more batches of 100 samples from the hook model using different values for the temperature (1.5 and 2.0), in order to ascertain how this might affect the score. Since the model only has 5 lost samples in the $t = 1.5$ model, we will use this as the benchmark set of samples for the model trained on the hook dataset, instead of the set produced by using $t = 1.0$, which had too many samples that could not be used by MelodyShape. I choose 1.5 and 2.0 as the temperatures empirically.

Indeed, as discussed in the section on the Sampling procedure, temperature adds a measure of randomness, which can alleviate an overfitted model. In this way, I argue that, in fact, the more relevant comparison would be between the folk model with $t = 1.0$ and the hook model with $t = 1.5$. I settle on the $t = 1.5$ batch as the benchmark for the hook model since it has very few `-inf` samples that could not be processed by the MelodyShape tool.

Even so, it seems like overall the plagiarism score is still higher for the model trained on the heterogeneous dataset. However, the higher SD means that the mean is not really meaningful, as already highlighted by Yang et. al.

One interpretation of this is that the model trained on the homogenous dataset (folk dataset) has learned how to both mimic the musical qualities of its dataset (see high overall OA inter-set distances in table 5.1) and, at the same time, the intrinsic patterns exhibited by the folk genre of music. It has learned to produce original recombinations of the patterns in the training set. This might be due to a large number of samples that it could learn from, as compared to the hook dataset.

On the other hand, the model trained on the heterogeneous dataset (hook) is suffering from a lack of originality perhaps due to a small number of samples from each of the genres present in the dataset, thus overfitting to a particular area of the function space, suffering from the problem known as *mode collapse*[2]. The problem here is due to both an uneven and a small dataset. The model needs more data in order to generalize better. Another problem could be that the hook dataset genres are also themselves different from one another, so we observe a reduced transfer learning potential from one genre to the other(s).

[2]This is a problem observed by Goodfellow in Generative Adversarial Networks (GANs). The network restricts its learning to a subset of the variety of samples [81].

The analogy here could be one between a musician that is aiming to become a very good musician within a certain genre and a musician that is aiming to be a very good musician but also *combine* genres in their writing. Intuitively, one musician could become very good at composing within a genre after studying and learning n songs in that genre. However, in order for a musician to become very good at m genres, they would need to study and learn $m \times n$ songs, n songs in each of the m genres. However, the hook dataset (comprised of multiple genres) only contains the same n number of songs as the folk dataset (in this case $n = 100000$, as discussed in the chapter on datasets). Also, n in my case represents the number of 4-bar phrases, which is not the equivalent of full songs.

Testing the hypothesis To test this hypothesis, I propose another experiment. I will train a model on the folk dataset using the same number of songs that the hook dataset has in a given genre.

By examining the hook webpage[3], I obtain a rough estimate of 1500 songs in the most popular genre in the data set, pop. These are usually made up of two sequences, one 8-bar sequence for the verse and one 8-bar sequence for the chorus. Each 8-bar sequence produces 5 4-bar sequences. We thus get 10 sequences for each song (5 from the chorus, 5 from the verse). In total, for one genre, we would have 15000 4-bar sequences.

I then train a model with the same architecture as above on a subset of the folk dataset of 15000 sequences. I generate 100 samples and compute the plagiarism scores. These can be observed in 5.5. As we see, the plagiarism score for a model trained on fewer examples is higher. This supports my hypothesis about the behaviour of the hook dataset when compared to the folk dataset.

Table 5.5: Experiment 1: Plagiarism scores for folk model trained on 15k sequences

model	**mean**	**SD**	**-inf**
folk(15k)	0.683	0.174	0

To confirm whether the number of samples has an impact on model originality I train a further set of models on different sized subsets of the folk dataset. The results can be seen in table 5.6 and in the fig. 5.2. We can see a clear trend in the decrease of the plagiarism scores as the model learns from a more ample dataset.

Table 5.6: Experiment 1: Plagiarism scores for folk model trained on different sizes

model	**mean**	**SD**	**-inf**
15k	0.683	0.174	0
30k	0.573	0.174	0
45k	0.278	0.222	0
60k	0.126	0.150	0

[3]I examine the pop genre, on https://www.hooktheory.com/theorytab/genres/pop. It is the genre with the most songs.

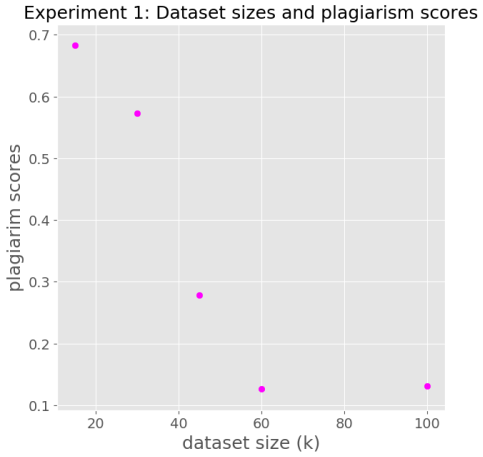
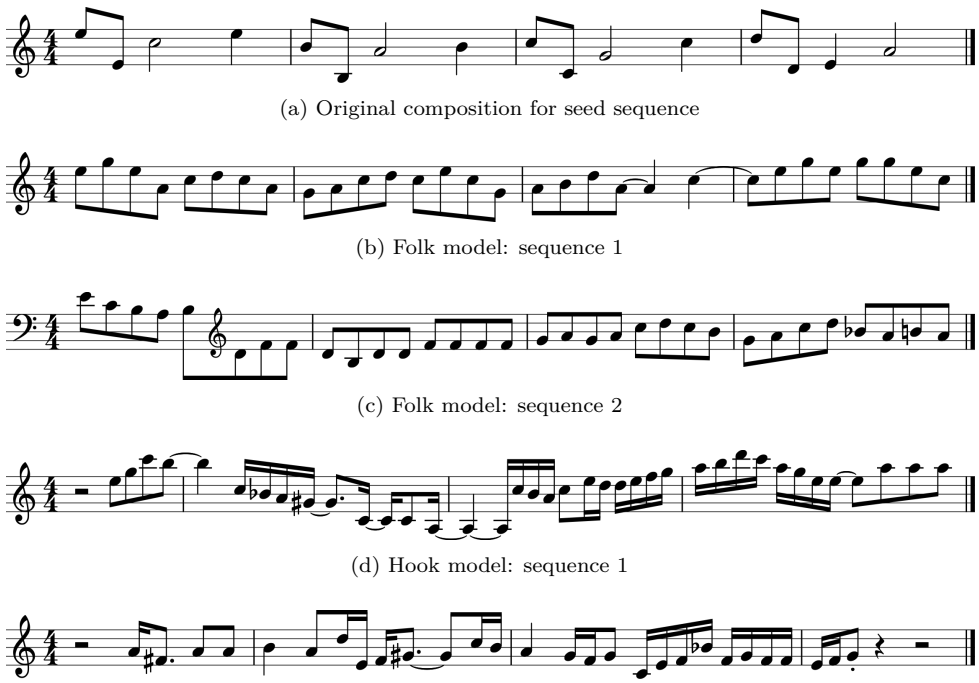


Figure 5.2: Scatter plot of plagiarism and dataset size

The more samples the system is exposed to, the wider vocabulary it can develop in composing new music in that style. A wider vocabulary can then lead to new and original ways of recombining existing patterns. By comparison, while the hook dataset does contain the same number of n patterns, these are difficult to combine together, since they are of different genres. Further research would be required on this topic, with a larger dataset of n(songs) $\times$ m(genres), in order to fully understand the learning process. As highlighted in the discussion on the data sets, finding a quality data set is one of the challenges in the field of symbolic music generation.

Subjective Analysis

In this part I will present two samples from each of the models and discuss them, in terms of how well they fit the seed sequence I have composed. We can see these in fig. 5.3.[4]



(a) Original composition for seed sequence

(b) Folk model: sequence 1

(c) Folk model: sequence 2

(d) Hook model: sequence 1

(e) Hook model: sequence 2

Figure 5.3: Experiment 1: Samples

[4]The treble clef symbol means that the following sequence should be played one octave higher.

Firstly, we can see that one of the hook samples ends prematurely. This is due to the network's tendency to simply repeat the last note (token) played. In this case, it's the `no new event` token. We can also notice that the hook samples do not exhibit as clear a melodic contour as the folk ones, being more random. This is due to the higher temperature of $t = 1.5$ that I have used to generate the samples. However, most of the samples in the $t = 1.0$ batch were unusable, consisting of one to four notes being held for long durations.

In the folk samples, we can observe a tendency for eighth notes rhythm. This indeed seems to confirm the observation that the folk model is characterized by a poor emulation of the training set's rhythmic qualities. The samples themselves present a musical melodic contour. The second sample in particular exhibits a substructure consisting of two sections, the first part in a lower register, and the second part ascending to a higher register. All of these samples seem to exhibit a problem of being stuck on the same note in different places.

Overall, I do not think these samples would work well as a continuation of the original seed sequence, as they do not preserve its rhythm. The folk samples at least present a modicum of melodic contour, though it does not fit with the seed sequence.

Experiment 2: dataset encoding

In this experiment, I train two models with the same architecture but with one difference: the encoding of the dataset. I then discuss how this (the independent variable) influences the characteristics of the datasets (the dependent variable).

As discussed in the section on dataset encodings I am focusing on two formats of encoding MIDI music. Firstly, the more common pianoroll format, and, secondly, the *melody* format proposed by the Magenta team in [17]. The former is more commonly employed in research but suffers from the "note ending" problem. The latter does not suffer from that limitation, but has not seen wide-scale experiments. I propose to compare how these two options influence learning.

Both the models are trained on the folk dataset.

The architecture is similar to the folk model from experiment 1, with the difference being in the input layer. This layer differs in the number of nodes, depending on the encoding used. The pianoroll yields a vocabulary of 56, while the *melody* has a vocabulary of 58. The difference is accounted for by the *melody* format using special tokens for `no new event` and padding.

Objective Analysis

In table 5.7 we can observe the statistics for the 100 samples generated with the model trained on the folk dataset encoded in the *melody* format. We notice that the system models the training system quite closely, with many of the absolute measures being very close to the training data: Pitch Count (PC), Pitch Interval (PI). This is also the case for intra-set measurements: Pitch Class Histogram (PCH), Pitch Class Transition Matrix (PCTM), and avg. inter-onset interval (IOI).

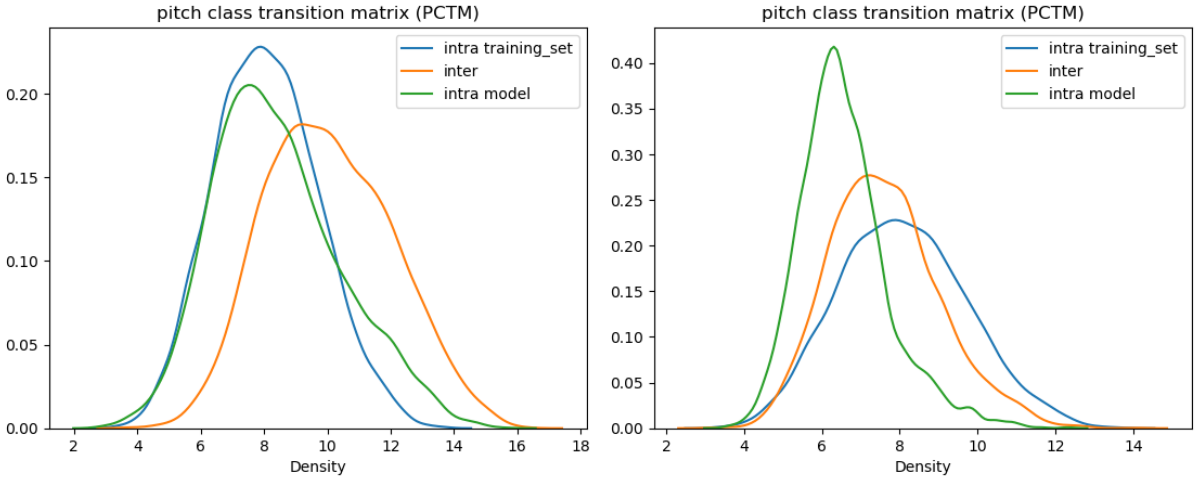
Table 5.7: Experiment 2: Training set (folk) and melody model comparison

	Training set (folk)				Model (melody)				Inter-set	
feat.	**mean**	**SD**	**intra-set mean**	**intra-set SD**	**mean**	**SD**	**intra-set mean**	**intra-set SD**	**KLD**	**OA**
PC	8.390	1.849	2.087	1.597	7.760	2.815	3.204	2.395	0.101	0.697
PR	14.110	3.493	3.935	3.026	15.050	7.384	8.226	6.518	0.032	0.603
PI	2.796	0.647	0.724	0.568	2.456	0.715	0.644	0.785	0.653	0.826
PCH	-	-	0.316	0.110	-	-	0.288	0.120	0.061	0.865
PCTM	-	-	8.081	1.645	-	-	8.351	2.012	0.033	0.915
pitch avg.	-	-	-	-	-	-	-	-	0.176	0.781
NC	28.070	3.567	3.976	3.147	30.300	3.273	2.217	4.089	0.948	0.544
IOI	0.286	0.042	0.043	0.042	0.271	0.084	0.035	0.115	4.736	0.609
NLH	-	-	0.354	0.264	-	-	0.080	0.124	0.965	0.266
NLTM	-	-	12.092	6.494	-	-	4.747	5.760	0.870	0.389
rhythm avg.	-	-	-	-	-	-	-	-	1.880	0.452
overall avg.	-	-	-	-	-	-	-	-	1.028	0.617

Table 5.8: Experiment 2: Training set (folk) and pianoroll model comparison

	Training set (folk)				Model (pianoroll)				Inter-set	
feat.	**mean**	**SD**	**intra-set mean**	**intra-set SD**	**mean**	**SD**	**intra-set mean**	**intra-set SD**	**KLD**	**OA**
PC	8.390	1.849	2.087	1.597	10.110	2.088	2.344	1.819	0.151	0.850
PR	14.110	3.493	3.935	3.026	17.600	4.675	5.203	4.134	0.024	0.822
PI	2.796	0.647	0.724	0.568	3.340	0.491	0.566	0.409	0.030	0.843
PCH	-	-	0.316	0.110	-	-	0.293	0.107	0.069	0.904
PCTM	-	-	8.081	1.645	-	-	6.567	1.146	0.328	0.551
pitch avg.	-	-	-	-	-	-	-	-	0.120	0.794
NC	28.070	3.567	3.976	3.147	23.750	3.103	3.506	2.676	0.052	0.902
IOI	0.286	0.042	0.043	0.042	0.347	0.054	0.057	0.050	0.104	0.817
NLH	-	-	0.354	0.264	-	-	0.183	0.088	0.326	0.648
NLTM	-	-	12.092	6.494	-	-	6.984	3.296	0.076	0.617
rhythm avg.	-	-	-	-	-	-	-	-	0.139	0.746
overall avg.	-	-	-	-	-	-	-	-	0.130	0.770

In 5.8 we observe the statistics from the model trained on the folk dataset encoded in pianoroll. We observe a large difference between how each model learns pitch features, as opposed to the rhythmic ones. The average OA for the rhythmic features is larger in the pianoroll model. However, the *melody* model seems better at learning melodic steps sequencing, as its OA and KLD are much larger, and lower, respectively, for the Pitch Class Transition Matrix (PCTM) feature, compared to the model trained with the pianoroll encoding. We can see the tendency of the *melody* model to be better at emulating the training set's melodic progression patterns in fig. 5.4. The graphs represent the distributions of intra-set and inter-set Euclidean distances of the PCTM feature. We can see that the *melody* model is indeed much closer to the training set's distribution.

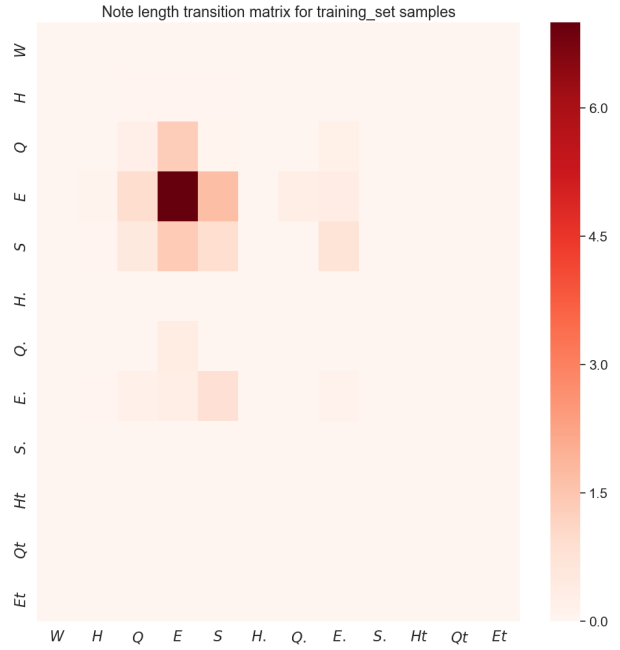


(a) PCTM for melody model

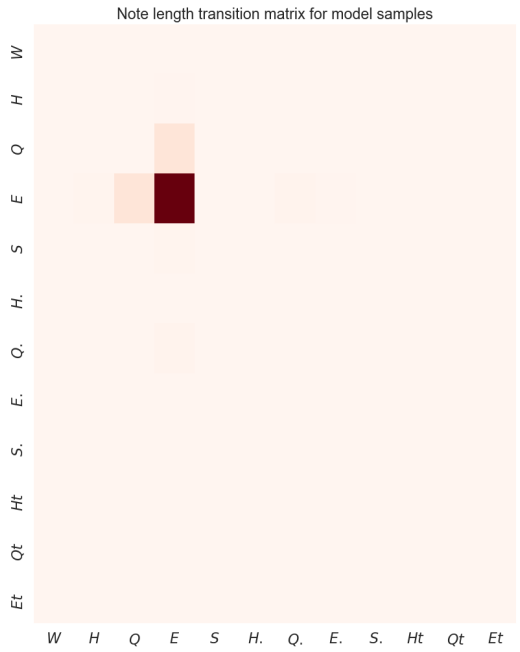
(b) PCTM for pianoroll model

Figure 5.4: Experiment 2: Pitch Class Transition Matrices (PCTM)

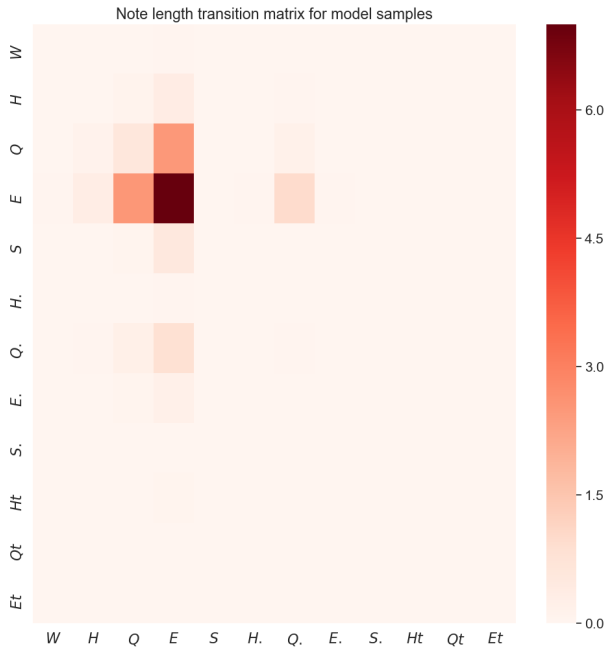
The biggest difference seems to be in the rhythmic features. The intra-set measurements for the Note Length Histogram (NLH) and Note Length Transition Matrix (NLTM) features are smaller in the *melody* model samples, compared to both the training set and the pianoroll set. This observation suggests that the model does not learn the entire variety of rhythmic expression, and instead reduces the vocabulary to a few rhythmic phrases. The much lower intra-set mean for the Note Length Transition Matrix (NLTM) means that the model "collapses" into a very small subset of rhythmic movements. Indeed, if we observe the NLTM heatmaps in fig. 5.5 we notice that the *melody* model in fig. 5.5b exhibits a denser concentration on fewer rhythmic transitions, compared to the pianoroll model, in fig. 5.5c, which exhibits more variety.



(a) NLTM for folk dataset



(b) NLTM for melody model dataset



(c) NLTM for pianoroll model dataset

Figure 5.5: Experiment 2: Note Length Transition Matrices. The labels represent, in order: Whole note, Half note, Quarter note, Eighth Note, Sixteenth Note, Dotted Half, Dotted Quarter, Dotted Eighth, Dotted Sixteenth, Triplet Half, Triplet Quarter, Triplet Eighth

In conclusion, we can say that the *melody* model seems to be better at modelling the melodic progressions of the datasets in terms of pitch (notes), while the pianoroll model seems to be overall better at capturing rhythmic patterns.

Plagiarism and originality

In table 5.9 we can see the plagiarism scores for the two models. It is immediately apparent that there is no noticeable difference between the two models in terms of originality and plagiarism. They both exhibit low scores of plagiarism. This further corroborates the observations from Experiment 1 that dataset size and stylistic consistency is a key aspect of producing such results, as both models in this experiment are trained on the same homogenous folk data set.

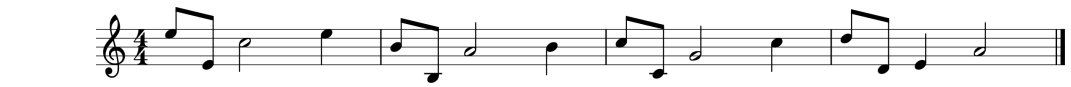
Table 5.9: Experiment 2: plagiarism scores for models

model	**mean**	**SD**	**-inf**
melody	0.113	0.057	0
pianoroll	0.140	0.060	0

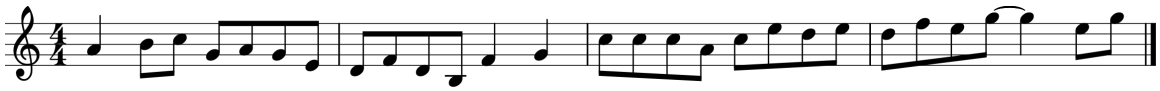
Subjective Analysis

In this part, I will present two samples from each of the models and discuss them, in terms of how well they fit the seed sequence I have composed and in terms of their own respective melody and rhythmic qualities. We can see these in fig. 5.6[5][6].

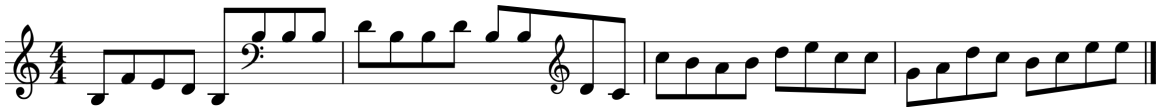
[5]The treble clef symbol means that the following sequence should be played one octave higher.
[6]The bass clef symbol means that the following sequence should be played one octave lower.



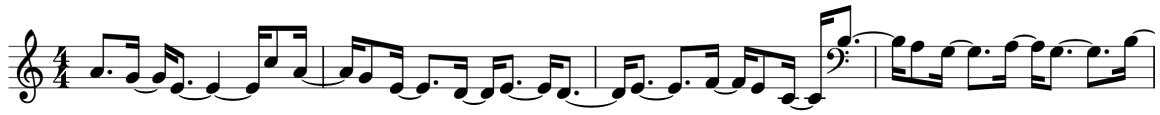
(a) Original composition for seed sequence

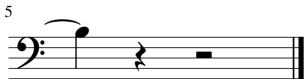


(b) melody model: sequence 1

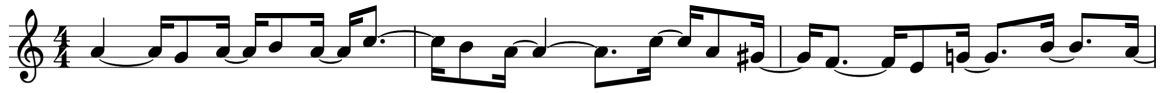


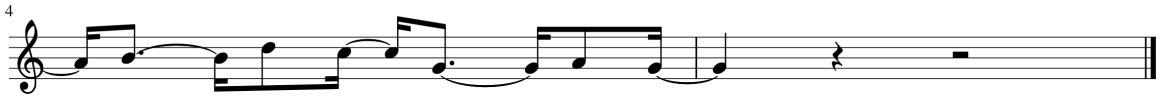
(c) melody model: sequence 2





(d) pianoroll model: sequence 1





(e) pianoroll model: sequence 2

Figure 5.6: Experiment 2: Samples

We immediately notice that the samples from the pianoroll model exhibit more rhythmic variety. This confirms the objective analysis, where we observed that the pianoroll model had a more diverse Note Length Transition Matrix (NLTM).

In terms of melodic quality, I think that the samples in the *melody* group present a structured sense of melodic progression, even though it does not fit the seed and it is not very original or captivating to the ear. On the other hand, the samples from the pianoroll group do not exhibit a contour that can be easily followed, but the melody itself feels more original. There is even a sense of chromaticism in the second sample, where we get a G# that is outside of the A minor key. It creates a tense and exotic feeling.

Overall, I would not say these samples fit the seed sequence very well either.

Experiment 3: attention mechanism

In this experiment, I compare two models based on the effect of adding an attention mechanism. I have defined attention in the state of the art and methods chapters (pg. 10 and pg. 19) as a mechanism that allows the models to learn what time steps are more relevant in predicting a specific next time

step. In this experiment having or not having an attention mechanism is the independent variable, and the dependent variable is the measures extracted from the generated samples. I aim to show that the attention mechanism can enhance the quality of the generated samples.

The architecture was as follows:

- input layer. 63 time steps with 58 vocabulary tokens
- 2 LSTM layers with 32 cells each, with a 40% dropout rate each.
- attention mechanism (for one of the models)
- output layer, 1 time step of 58 dimensions.

The model was trained on the folk dataset with the *melody* encoding, with the Adam optimizer, for 50 epochs, with a mechanism for reducing the learning rate on plateaus.

Objective Analysis

We can see the results for the model without attention (the baseline) in table 5.10.

Table 5.10: Experiment 3: Training set (folk) and model (without attention)

	Training set				Model (without attention)				Inter-set	
feat.	**mean**	**SD**	**intra-set mean**	**intra-set SD**	**mean**	**SD**	**intra-set mean**	**intra-set SD**	**KLD**	**OA**
PC	8.390	1.849	2.087	1.597	7.720	1.477	1.591	1.370	0.080	0.761
PR	14.110	3.493	3.935	3.026	14.040	3.580	3.897	3.272	0.061	0.846
PI	2.796	0.647	0.724	0.568	2.983	0.473	0.536	0.407	0.005	0.825
PCH	-	-	0.316	0.110	-	-	0.260	0.083	0.020	0.787
PCTM	-	-	8.081	1.645	-	-	7.371	1.215	0.095	0.764
pitch avg.	-	-	-	-	-	-	-	-	0.052	0.797
NC	28.070	3.567	3.976	3.147	28.520	1.841	2.061	1.612	0.185	0.626
IOI	0.286	0.042	0.043	0.042	0.279	0.018	0.020	0.016	0.299	0.658
NLH	-	-	0.354	0.264	-	-	0.094	0.058	0.237	0.345
NLTM	-	-	12.092	6.494	-	-	5.374	3.299	0.169	0.497
rhythm avg.	-	-	-	-	-	-	-	-	0.223	0.531
overall avg.	-	-	-	-	-	-	-	-	0.137	0.664

We can see the results for the model with the attention mechanism in table 5.11.

Table 5.11: Experiment 3: Training set (folk) and model (with attention)

	Training set (folk)				Model (with attention)				Inter-set	
feat.	**mean**	**SD**	**intra-set mean**	**intra-set SD**	**mean**	**SD**	**intra-set mean**	**intra-set SD**	**KLD**	**OA**
PC	8.390	1.849	2.087	1.597	10.220	1.616	1.809	1.416	0.032	0.832
PR	14.110	3.493	3.935	3.026	22.650	5.825	6.377	5.280	0.088	0.739
PI	2.796	0.647	0.724	0.568	5.849	1.245	1.418	1.059	0.009	0.669
PCH	-	-	0.316	0.110	-	-	0.277	0.084	0.005	0.840
PCTM	-	-	8.081	1.645	-	-	6.514	0.963	0.378	0.504
pitch avg.	-	-	-	-	-	-	-	-	0.102	0.717
NC	28.070	3.567	3.976	3.147	22.930	1.620	1.809	1.424	0.221	0.545
IOI	0.286	0.042	0.043	0.042	0.348	0.024	0.026	0.022	0.123	0.750
NLH	-	-	0.354	0.264	-	-	0.131	0.076	0.184	0.468
NLTM	-	-	12.092	6.494	-	-	4.497	1.696	0.232	0.315
rhythm avg.	-	-	-	-	-	-	-	-	0.190	0.519
overall avg.	-	-	-	-	-	-	-	-	0.146	0.618

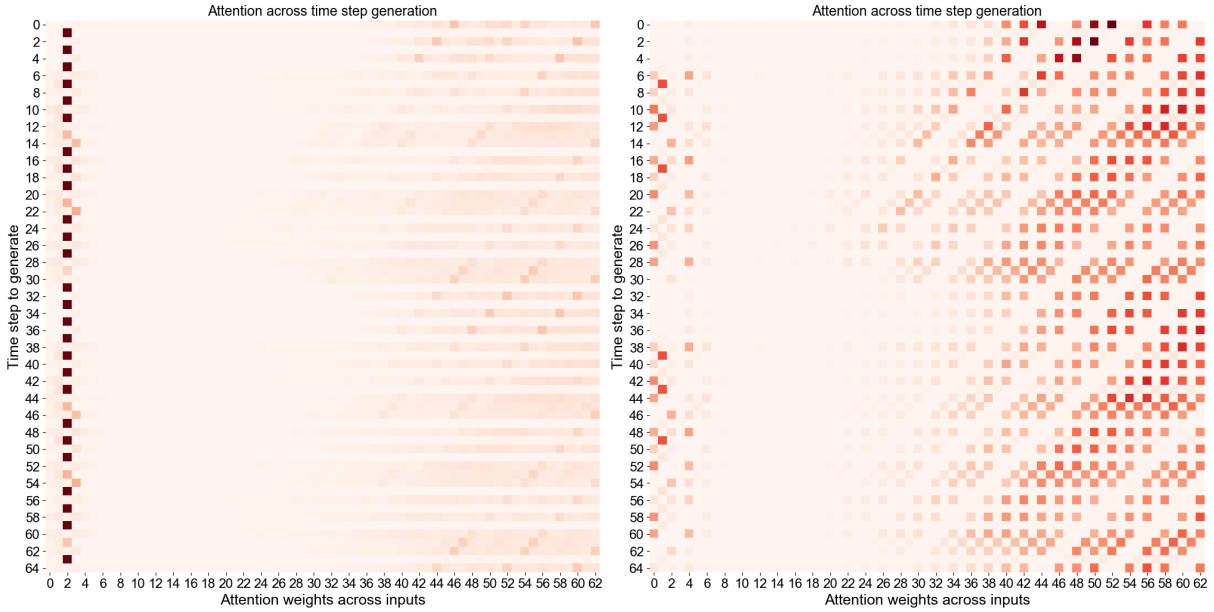
We can see that the attention model exhibits a higher divergence from the training set, with a lower overall OA for most features. As we will see from the subjective analysis, this can be attributed to the

network's tendency to emulate the melody contour of the seed sequence. Thus, the generated samples do not, overall, fit the statistics of the training set.

There is a small improvement in terms of OA for the Pitch Class Histogram (PCH) for the attention-based model. However, this is weighed down by the overall average for the OA.

To explore the effects of the attention mechanism I present in fig. 5.7a a heatmap of the attention weights across the generated time steps. This can be read as, e.g. "when generating time step 64 the attention mechanism was particularly attending the time steps 44, 52, 60 in the input sequence"[7]. We can observe a pattern on the right side of the heatmap (from input time step 38 onwards), that advances in relation to the newly generated time steps. It seems like the attention mechanism is focused on a specific distance d between the time step index of the input sequence and the time step index of the generated sequence. Thus, we have $y = x + d$, where d is the distance between output step y and the attended input step x. Since we have multiple indices at each generated time step that seem to have a high heat map weight (e.g. for the generated time step 0 the model is attending indices 46, 54, 62), we can say that d is a set of distances. Then x is a set of time step indices in the input. The time steps $x = y - d$ are the focus of the attention mechanism in deciding which new note (event) to generate.

We can also notice that there is another alternating pattern: the attention is either focused on the set x I just mentioned or on the second time step in the input. The weight associated with this second time is also very large, as we can see. From further investigations I observe that most of the large attention weights are pointing towards the `no new event` token[8]. In order to obtain a clear image of how the attention mechanism is working, I mask all the attention weights that are pointing to time steps with this token. I then create the heatmap of the attention that can be seen in fig. 5.7b. In this figure the pattern of the attention mechanism is more apparent.



(a) Visualization of attention weights. The max weight factor has been clipped to 0.4 in order to better highlight the progressing pattern on the right side.

(b) Visualization of attention weights. In this case I have removed all attention weights that were associated with the `no new event` token

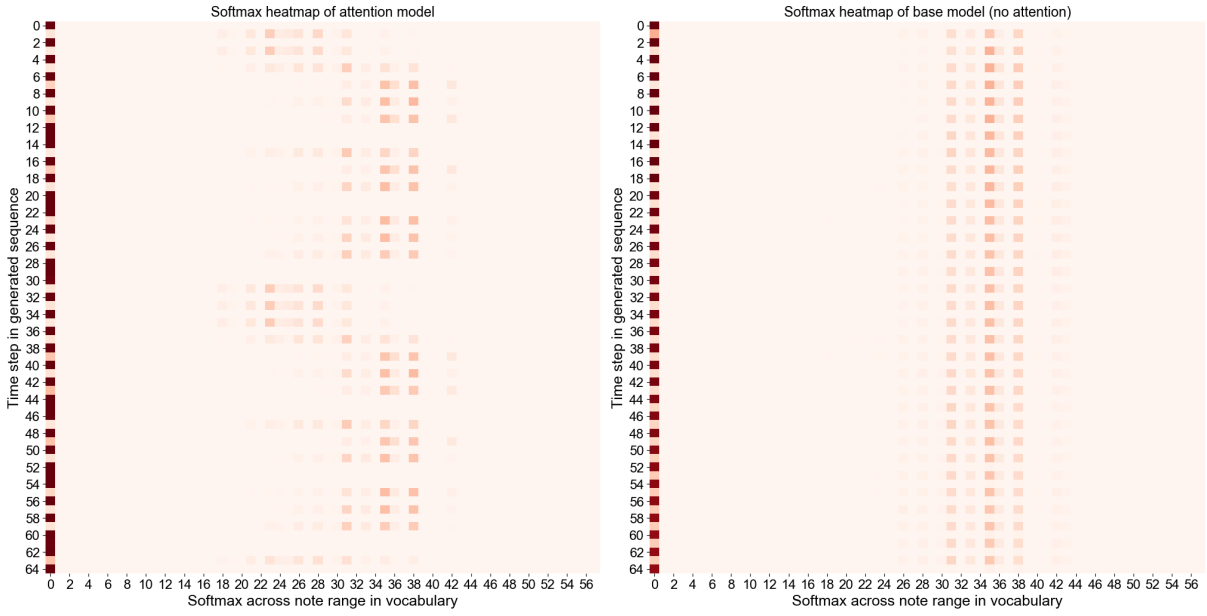
Figure 5.7: Attention weights

[7] In generating these heatmaps I have concatenated all the attention weights across the newly generated 65 time steps. Of these, the first time step is part of the four bars of the training sequence, being its last time step (the one we train the model to predict). See discussion in sampling procedure (pg. 20).

[8] In the *melody* encoding this token means that the previously generated note is held for the duration.

We notice in fig. 5.7b that there is an area in the input time steps that are not attended almost at all during generation (approx. from index 6 to index 20). It is not clear why the attention mechanism does not focus on this area. One interpretation could be that the model relies on the information from the last half of the input and the first several indices. It is overall difficult to ascertain why the attention mechanism converges on this specific set of distances, but the consistency of the pattern supports the hypothesis that the attention mechanism is guiding the model to focus on a particular set of input time steps that it deems relevant in deciding the new outputs.

In order to further examine the behaviour of the network, I provide the visualization of the softmax probability distributions. In fig. 5.8 we can see the softmax probabilities for the attention and baseline model (without an attention mechanism). We notice that in the attention model (fig. 5.8a) the pattern of probabilities alternates in an almost regular pattern, while in the base model there is no such alternation and the probabilities are distributed in the same way at every time step. We also observe that both models exhibit a very high probability weight on the first element of the note range, in alternation with the more spread distribution in the centre-right area (30-38 for baseline, 22-38 for attention model). This first element is not an actual note, but the `no new event` token. The regular pattern in the attention model also supports the hypothesis that the attention mechanism helps the model develop a better underlying long-term structure in the generated pieces.



(a) Softmax probabilities for attention model

(b) Softmax probabilities for base model (no attention)

Figure 5.8: Experiment 3: Softmax probability distributions across note range

Plagiarism and originality

In table 5.12 we can see the results for the plagiarism scores for the two models. We can see that they both exhibit low plagiarism, along with a low standard deviation.

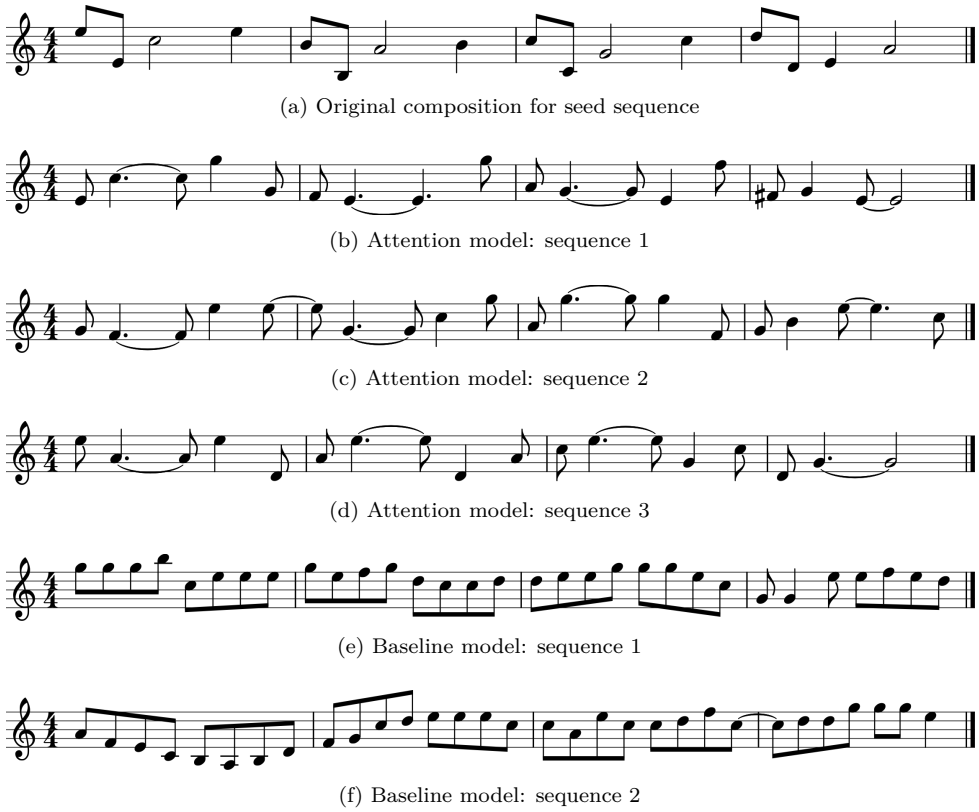
Table 5.12: Experiment 3: plagiarism scores

model	mean	SD	-inf
attention	0.061	0.048	0
no attention	0.098	0.040	0

It seems like objectively the attention mechanism does not help in my experiment. Further experiments would be required to further solidify or refute this claim. In the case of Google Magenta's attention model [17], we observe a definite improvement in the quality of the samples. However, they also use a special token symbolizing the repetition of a previous musical phrase. As we will see from the subjective analysis, my model does preserve the overall structure and outline of the seed melody.

Subjective Analysis

In this part, I will present samples from each of the models and discuss them, in terms of how well they fit the seed sequence I have composed. We can see these in fig. 5.9.



(a) Original composition for seed sequence

(b) Attention model: sequence 1

(c) Attention model: sequence 2

(d) Attention model: sequence 3

(e) Baseline model: sequence 1

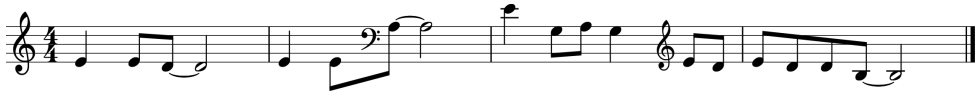
(f) Baseline model: sequence 2

Figure 5.9: Experiment 3: Samples

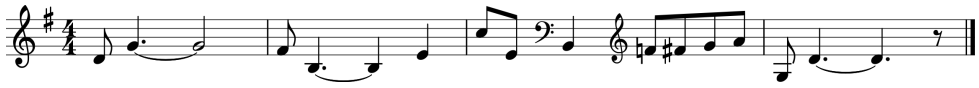
We immediately observe that the attention model preserves the structure of the seed sequence much better than the baseline model does. It seems like the attention mechanism does improve the ability of the network to preserve the long-term structure. We also notice that the baseline is again fixated on a rhythm of eighth notes.

In order to further highlight the attention mechanism's ability to preserve structure I compose another seed sequence and generate another set of samples, in fig. 5.10[9][10]:

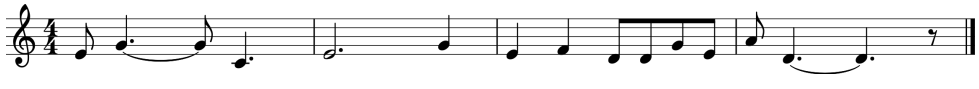
[9]The treble clef symbol means that the following sequence should be played one octave higher.
[10]The bass clef symbol means that the following sequence should be played one octave lower.



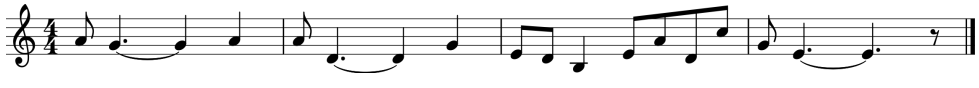
(a) Original composition for seed sequence



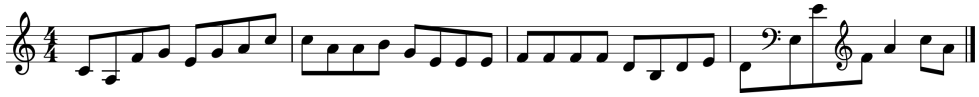
(b) Attention model: sequence 1



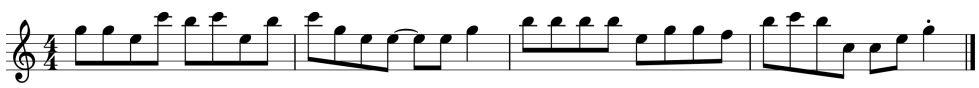
(c) Attention model: sequence 2



(d) Attention model: sequence 3



(e) Baseline model: sequence 1



(f) Baseline model: sequence 2

Figure 5.10: Experiment 3: Second seed sequence and samples

Again, we observe that the attention model surpasses the baseline in capturing a sense of melodic structure. In particular, sequence 3 compliments the seed sequence well, in the rhythmic patterns in the third and fourth bar.

Subjectively, I think some of these samples could actually be used in a musical composition, as a continuation or variation of the initial idea. It seems that the attention layer does improve the quality of the samples. This is not apparent in the objective analysis. One explanation for this would be that the model is focused on replicating the structure of the seed sequence to the detriment of the patterns in the original training set, and thus the generated batch of samples does not resemble the training set as much.

Experiment 4: bidirectionality

In this experiment, I aim to test the effects of a LSTM bidirectional wrapper on the quality of the results. For this, I propose to compare a model that does not have a bidirectional wrapper (the baseline) with a model that does. The model without bidirectionality has the same number of parameters in total as the model with bidirectionality. A bidirectional wrapper doubles the number of cells in a layer, by adding another layer of cells that process the input sequence in reverse order. Thus, if we are to mark as $2 \times n$ the number of cells in a layer, I compare the effect of converting n of those cells into reverse order LSTM cells, as opposed to having more of them focus on the chronological order.

The architectures are as follows:

- input layer: 63 time steps, at 58 dimensions each. The models are built on top of the *melody* encoding of the folk dataset

- 2 layers, each with 64 LSTM cells, and a 40% dropout rate. The difference is that in the case of one of the models 32 of these cells consider the input in reverse order
- output layer: 1 time step of 58 dimensions.

The models are trained with the Adam optimizer, for 50 epochs, with a mechanism for reducing learning rate on plateaus.

Objective Analysis

We can observe the quantitative evaluation for the bidirectional model in table 5.13. Comparing this with the number for the baseline model (table 5.14) we observe that bidirectionality improves the performance of the model across multiple features. The most striking differences are in the Pitch Class Histogram (PCH, Note Count (NC), Inter-onset Interval (IOI), based on the inter-set OA.

Even in terms of absolute values, we can see that the model with the bidirectionality wrapper is better at emulating the patterns of the training set. The mean and SD values for the Pitch Range (PR), Pitch Interval (PI) are closer to the training data.

Table 5.13: Experiment 4: Comparison between training set (folk) and model (with bidir.)

	Training set (folk)				**Model (with bidir.)**				**Inter-set**	
feat.	**mean**	**SD**	**intra-set mean**	**intra-set SD**	**mean**	**SD**	**intra-set mean**	**intra-set SD**	**KLD**	**OA**
PC	8.390	1.849	2.087	1.597	7.380	1.928	2.162	1.683	0.155	0.885
PR	14.110	3.493	3.935	3.026	14.230	5.435	5.651	5.266	0.062	0.789
PI	2.796	0.647	0.724	0.568	2.681	0.679	0.716	0.648	0.063	0.895
PCH	-	-	0.316	0.110	-	-	0.321	0.121	0.031	0.954
PCTM	-	-	8.081	1.645	-	-	7.559	1.811	0.320	0.802
pitch avg.	-	-	-	-	-	-	-	-	0.126	0.865
NC	28.070	3.567	3.976	3.147	25.540	3.278	3.661	2.883	0.041	0.915
IOI	0.286	0.042	0.043	0.042	0.316	0.046	0.049	0.044	0.088	0.875
NLH	-	-	0.354	0.264	-	-	0.153	0.090	0.112	0.574
NLTM	-	-	12.092	6.494	-	-	7.449	4.224	0.040	0.687
rhythm avg.	-	-	-	-	-	-	-	-	0.070	0.763
overall avg.	-	-	-	-	-	-	-	-	0.098	0.814

Table 5.14: Experiment 4: Comparison between training set (folk) and model (without bidir.)

	Training set (folk)				**Model (no bidir.)**				**Inter-set**	
feat.	**mean**	**SD**	**intra-set mean**	**intra-set SD**	**mean**	**SD**	**intra-set mean**	**intra-set SD**	**KLD**	**OA**
PC	8.390	1.849	2.087	1.597	9.340	1.779	2.019	1.521	0.234	0.894
PR	14.110	3.493	3.935	3.026	18.950	4.955	5.600	4.270	0.028	0.780
PI	2.796	0.647	0.724	0.568	4.008	0.758	0.854	0.657	0.003	0.891
PCH	-	-	0.316	0.110	-	-	0.242	0.082	0.040	0.709
PCTM	-	-	8.081	1.645	-	-	7.267	1.184	0.092	0.727
pitch avg.	-	-	-	-	-	-	-	-	0.079	0.800
NC	28.070	3.567	3.976	3.147	28.500	1.987	2.241	1.720	0.080	0.667
IOI	0.286	0.042	0.043	0.042	0.280	0.020	0.022	0.017	0.256	0.693
NLH	-	-	0.354	0.264	-	-	0.102	0.060	0.247	0.380
NLTM	-	-	12.092	6.494	-	-	5.840	3.507	0.059	0.550
rhythm avg.	-	-	-	-	-	-	-	-	0.160	0.573
overall avg.	-	-	-	-	-	-	-	-	0.120	0.686

Plagiarism and originality

We can see in table 5.15 that even though the model with the bidirectionality wrapper is better at emulating the training set, it does not do so by simply replicating the results. Its plagiarism score is still

low and close to the model without bidirectionality.

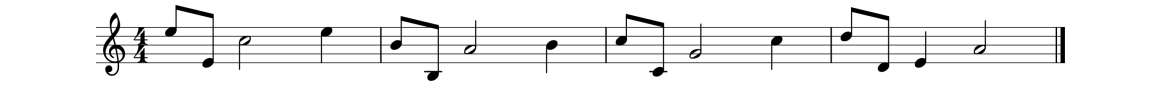
Table 5.15: Experiment 4: Model plagiarism scores

model	mean	SD	-inf
bidir	0.121	0.042	0
simple	0.091	0.048	0

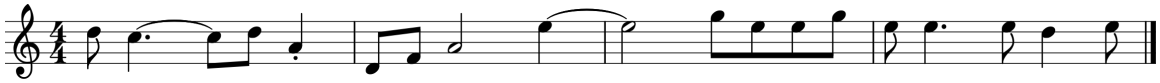
It seems like adding a bidirectional wrapper has overall helped the model to better learn the patterns related to both pitch and rhythm.

Subjective Analysis

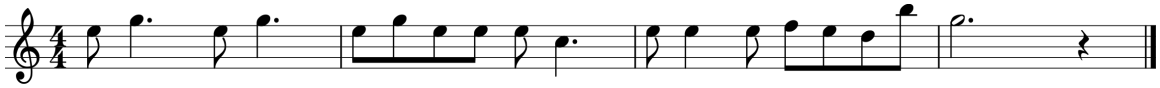
In this part, I will present two samples from each of the models and discuss them, in terms of how well they fit the seed sequence I have composed and in terms of their own respective melody and rhythmic qualities. We can see these in fig. 5.11.



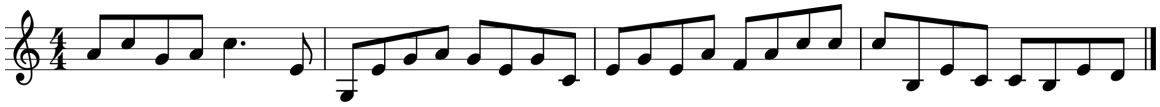
(a) Original composition for seed sequence



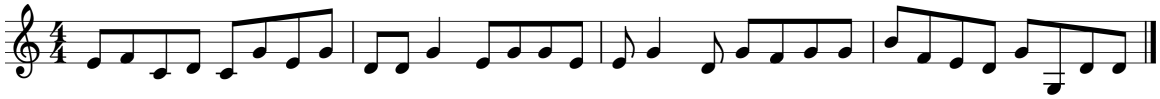
(b) bidirectional model: sequence 1



(c) bidirectional model: sequence 2



(d) baseline model: sequence 1



(e) baseline model: sequence 2

Figure 5.11: Experiment 4: Samples

We can see that the baseline model is again stuck in the eighth note rhythm, even though the samples present a pleasant overall melody. On the other hand, the model with bidirectionality produces pieces that seem to better resemble the melodic structure of the input composition.

Overall, I would say that using a bidirectional wrapper instead of doubling the number of LSTM cells improves the quality of the samples, with respect to how they continue the original input.

User evaluation

For the qualitative assessment of the samples, I employ a user study on the internet and reach out to people in my social circle.[11] The purpose of this survey was to assess the overall "musicality" of the samples generated by my proposed model. Since this is a very subjective topic, a user study was required. I compare the samples with original samples from the training set.

The users are shown 10 samples of 4 bars each from the model, mixed with 10 random samples from the folk training set. They are then asked to answer the following two questions for each sample:

1. *Can you tap along to the piece?* This question is meant to evaluate the rhythmic consistency of the melody. This can be interpreted in multiple ways, as we will see in the analysis of the results. However, attempting to tap along to the piece is one of the ways in which the listener can sync rhythmically to the piece [82, p. 466].
2. *How musically pleasing do you find the melody?* This question targets the overall aesthetic impact of the melody itself.

I chose to frame the question in a subjective manner so as to not enforce any specific understanding of what music should be. Since it is a strictly subjective evaluation, I want to evaluate what users think of the pieces without any specific preconceived notions.

The answers are on a 1 to 5 Likert scale. I also ask them to grade their own musical experience and background, also on a 1 to 5 scale. The options here are:

1. Have no knowledge/don't practice an instrument.
2. Some knowledge of music theory/practice a musical instrument once in a while
3. Years of experience playing an instrument/solid knowledge of music theory/in an amateur band
4. I am a professional musician/music teacher - years of practice and sophisticated knowledge
5. I am a professional composer/producer/songwriter or scholar/researcher

This question is meant to provide an evaluation of the musical background of the users. The answers from a user with more musical experience would carry more weight in the evaluation.

In total, I have received 33 answers. In fig. 5.12 we can see the distribution of user according to their expertise levels. 0 is the lowest, 4 is the highest.

[11] The survey can be viewed at https://forms.gle/CyG5qkVvCj4Puox78

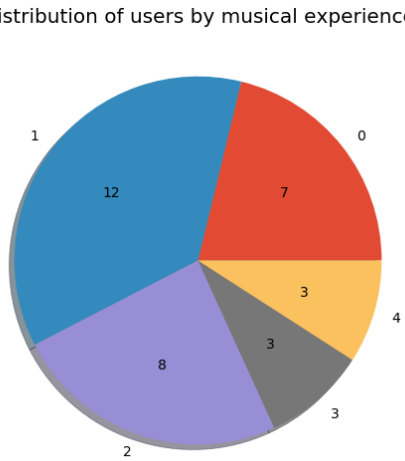


Figure 5.12: Distribution of users according to expertise levels

The model trained in this case had the following architecture:

- folk dataset in *melody* encoding format (58 vocabulary tokens), in 63 time steps
- 2 layers of 64 cells each, with a bidirectional wrapper on every layer. I also applied a 40% dropout rate to each of the layers
- attention layer on top of the second layer
- output layer with 1 time step and 58 units

The model is trained for 50 epochs, using an Adam optimizer. As seed sequence, I have used "Mary had a little lamb". This fits the genre of folk music, thus allowing the model to produce higher quality samples for the genre it was trained on.

A similar method of user-study is employed by Dong et. al in [73] and by Huang et. al in [70]. They also use a Likert scale and pose an open question that can be interpreted subjectively.

Results and discussion

I have decided to consider the results on an aggregate by experience level (levels 0 and 1 in the group labelled "lower", and levels 2,3, and 4 in the group labelled "upper"). In this way, I can observe if there is a difference in the assessment of the samples by user expertise. I have chosen to group these specific levels because in this clustering they form an approximately equal number of users (19 in the first, 14 in the second) but still maintain a grouping based on expertise.

In the tables 5.16 and 5.17 we can see the statistics. We can also see the results in the bar plot in fig. 5.13 and fig. 5.14 (the vertical bar represents the standard deviation of that group).

Table 5.16: User study results (in general)

	mean	SD
train rhythm	4.227	0.934
gen. rhythm	3.894	1.156
train melody	3.606	1.040
gen. melody	3.030	1.018

Table 5.17: User study results (aggregate per expertise levels)

	mean	SD
lower train rhythm	4.042	0.967
lower gen. rhythm	3.563	1.224
lower train melody	3.668	1.086
lower gen. melody	3.147	1.031
upper train rhythm	4.479	0.823
upper gen. rhythm	4.343	0.876
upper train melody	3.521	0.967
upper gen. melody	2.871	0.977

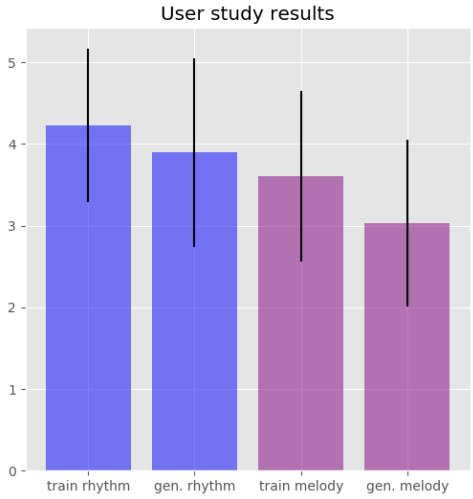


Figure 5.13: User study results (in general)

From fig. 5.13 we can see that there is a difference between the training set (the original music used to train the model) and the samples produced by the model, in terms of perceived quality. This is reflected in the rhythmic section, but also in the melodic section. The latter is also more pronounced. This means that overall the users did not think the model's samples were as musical as the samples from the original set, from their own subjective perspectives.

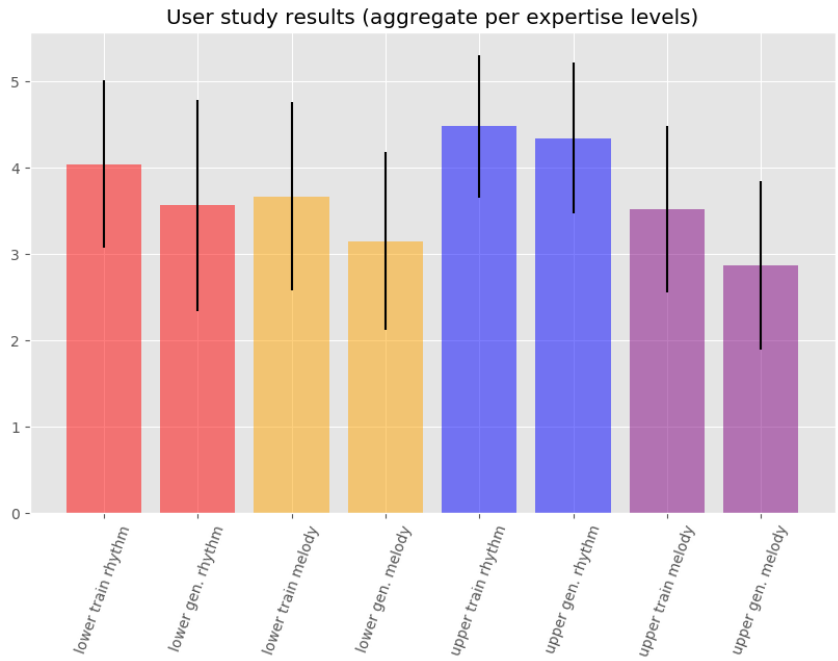


Figure 5.14: User study results (aggregate per expertise levels)

In fig. 5.14 we can see that the group with less experience (on the left, in red and yellow) perceives the rhythmic qualities of both the training and the generated samples as less pleasant overall, when compared to the scores received by the more musically experienced users. One interpretation of this could be that these users have a smaller vocabulary of rhythmic patterns, and thus do not perceive the rhythms as coherent or enjoyable. Traditional folk music is not popular with mainstream audiences. On the other hand, more experienced musicians might have come in contact with more rhythmically diverse songs, either in their studies, research, or composition. The larger standard deviation in the rhythmic scores in the 'lower' group makes the mean less significant, so it is difficult to conclude anything definite from the results. The standard deviation for the rhythmic evaluation in the 'upper' group is also smaller, making that score more reliable.

In terms of melodic appeal, it seems that the more experienced users found the generated melodies overall less pleasant, when compared to the 'lower' group. We can see that the scores for the training melodies are about the same value for both the 'upper' and the 'lower' group.

Overall, it is difficult to make any sound conclusions based on the limited number of answers and the small number of answers from more experienced musicians. The overall verdict would be that the generated samples are less musically pleasant for users, with rhythm being slightly better perceived than melody.

6 | Conclusions

In this work, I have addressed two main hypotheses in the topic of symbolic music generation with machine learning. Firstly, I have posed the question of how a dataset might affect the quality of a model's generated samples. I have tackled this by experimenting with two datasets that differed in terms of stylistic homogeneity (the folk and hook datasets), and by experimenting with two different dataset encoding formats (pianoroll and *melody*).

Secondly, I have experimented with how deep learning methods for sequential learning help improve the learning process. I explored this by performing two experiments, one for the attention mechanism [18] and for the bidirectional mechanism [21].

I have answered these questions with a methodology involving both quantitative and qualitative methods. In terms of the former, I have employed the methods proposed by Yang et al. in [13], focusing on a domain knowledge statistical analysis of the generated samples with comparison to the training dataset. For a quantitative assessment, I have also investigated the originality of the model's output, using the MelodyShape algorithm [14]. In terms of qualitative analyses, I have performed a user study where I have asked users to rate the generated music and the training samples. I have also performed a subjective assessment of the model by attempting to use it as a musician would. I have thus written an original piece of music to be used as a seed sequence and then evaluated the quality of the output samples.

As conclusions, I have observed that using a homogenous dataset (folk) yields more musically pleasing results, both quantitatively and subjectively. From further experiments, I noticed that this is due to the larger number of songs that the model can learn from. On the other hand, in the heterogenous multi-genre dataset (hook) of the same size, the model had problems learning and generating new material. This was due to the limited number of songs per genre.

I have also inferred that dataset encoding does not result in any major differences. The pianoroll model did present a slight improvement in terms of rhythmic variety over the *melody* system.

In experimenting with the attention mechanism I observe a substantially better melodic structure in relation to the seed sequence, in the subjective analysis. The objective analysis does not reflect an improvement. My interpretation of this was that the attention system seeks to emulate the seed sequence to the detriment of the training set. Thus, this does not reveal itself in the overall statistics.

In the fourth experiment, on bidirectionality, I observe an improvement both in the quantitative and the subjective analyses from converting half of the cells of the LSTM layers into LSTM cells processing the negative time direction.

Further research and limitations

There are multiple paths for further research. One would be to develop higher quality datasets. These datasets should, first of all, contain a larger amount of samples, include a more diverse range of genres, and also the metadata about the songs themselves (genre, key, year etc.). This would allow for experimenting with different constraints and observing the difference in results.

On the topic of datasets, the project would also benefit from experimenting with dataset augmentation by key transposition. As I have already noted, in this project I have opted to transpose all the songs to the key of C major (or A minor). I would be curious to analyze how the system would fare with a dataset where all the songs are transposed in all the keys.

Still on the area of datasets and dataset preprocessing, I would also propose to attempt a project with a more granular quantization of the encoding formats. In the current work, I choose to encode a bar of music as 16 steps (4 steps per quarter note). However, this does not capture triplet notes, which could have added a layer of rhythmic variety to the samples.

I would have also liked to experiment with a relative pitch encoding of the MIDI files. In this case, notes would not be represented as absolute pitch values, but as relative steps from the initial note. In this encoding, there also would not be a need for transposing the samples across keys, as each encoded sample could be interpreted as being in any key.

Another improvement would be to employ a measure of user constraints at generation time. This would fit under the **control** challenge, as catalogued in [23] by Briot and Pachet. One way of doing this, as proposed in [50] would be to impose specific musical pitches at specific time steps. Thus the model would be forced to adapt and include the user's input into the generation process. Combining that approach with the attention mechanism would be a worthy avenue of exploration.

It could also be worth to investigate how longer sequences of music develop. In my model I use 4 bars as the input length. This limits the scope of the attention mechanism and RNN internal state to only the past 4 bars. Experimenting with longer sequences (e.g. 8 bars) might be worth considering in order to improve the long-term structure of a melody.

In terms of a subjective evaluation, the experiments would be more useful if other more experienced musicians could experiment with the model and provide their own thoughts on how the samples matched their original input compositions. My expertise is limited and I might be biased in favour of the system.

In the current work, I combat overfitting and plateaus by employing a mechanism of decreasing the learning rate. However, this still leaves room for some overfitting, as was seen in the hook model in experiment 1. Further work would also apply an early stopping system, ensuring that the model does not overfit the training data.

Even though Yang and Lerch propose Kullberck-Leibler Divergence (KLD) as a measure of difference between two distributions, this is, as discussed, not a symmetric or bounded metric (see page 22). As an alternative, the quantitative evaluation could be enhanced to use the *Earth-Mover* (Wasserstein) distance. As discussed in [83] the EM distance has been proven as an improvement on KLD.

Finally, even though in this paper I choose to restrict myself to monophonic melodies, the experiments could also be applied to polyphonic music. These could either be polyphonic counter-melodies, or a dataset composed of chords and melodies. This could perhaps lead to more harmonically pleasing results, as the systems would have more information to learn from. It would thus be able to produce higher quality results.

References

[1] D. Epstein, "Beyond orpheus: Studies in musical structure," 1980.

[2] E. W. Marvin and R. Hermann, *Concert music, rock, and jazz since 1945: Essays and analytical studies*, vol. 2. University Rochester Press, 2002.

[3] S. Jarrett and H. Day, *Music composition for dummies*. John Wiley & Sons, 2008.

[4] M. C. Mozer, "Neural network music composition by prediction: Exploring the benefits of psychoacoustic constraints and multi-scale processing," *Connection Science*, vol. 6, nos. 2-3, pp. 247–280, 1994.

[5] J. J. Bharucha and P. M. Todd, "Modeling the perception of tonal structure with neural nets," *Computer Music Journal*, vol. 13, no. 4, pp. 44–53, 1989.

[6] Google, "Celebrating johann sebastian bach doodle" [Online]. Available: https://www.google.com/doodles/celebrating-johann-sebastian-bach

[7] C.-Z. A. Huang, T. Cooijmnas, A. Roberts, A. Courville, and D. Eck, "Counterpoint by convolution," *ISMIR*, 2017.

[8] F. Liang, "Bachbot: Automatic composition in the style of bach chorales," *University of Cambridge*, vol. 8, pp. 19–48, 2016.

[9] X. F. Liu, K. T. Chi, and M. Small, "Complex network structure of musical compositions: Algorithmic generation of appealing music," *Physica A: Statistical Mechanics and its Applications*, vol. 389, no. 1, pp. 126–132, 2010.

[10] G. Hadjeres, F. Pachet, and F. Nielsen, "Deepbach: A steerable model for bach chorales generation," in *Proceedings of the 34th international conference on machine learning-volume 70*, 2017, pp. 1362–1371.

[11] J.-P. Briot, G. Hadjeres, and F. Pachet, "Deep Learning Techniques for Music Generation - A Survey," *arXiv:1709.01620 [cs]*, Sep. 2017 [Online]. Available: http://arxiv.org/abs/1709.01620. [Accessed: 21-Mar-2019]

[12] A. van den Oord *et al.*, "Wavenet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499*, 2016.

[13] L.-C. Yang and A. Lerch, "On the evaluation of generative models in music," *Neural Computing and Applications*, Nov. 2018 [Online]. Available: http://link.springer.com/10.1007/s00521-018-3849-7. [Accessed: 20-Mar-2019]

[14] J. Urbano, "MelodyShape: A library and tool for symbolic melodic similarity based on shape similarity," *Online: https://github. com/julian-urbano/MelodyShape*, 2013.

[15] J. Urbano, J. Lloréns, J. Morato, and S. Sánchez-Cuadrado, "Melodic similarity through shape similarity," in *International symposium on computer music modeling and retrieval*, 2010, pp. 338–355.

[16] A. Roberts, J. Engel, C. Raffel, C. Hawthorne, and D. Eck, "A hierarchical latent vector model for learning long-term structure in music," *arXiv:1803.05428 [cs, eess, stat]*, Mar. 2018 [Online]. Available: http://arxiv.org/abs/1803.05428. [Accessed: 20-Mar-2019]

[17] D. R. Waite E.; Eck and D. Abolafia, "Generating long-term structure in songs and stories," 2016 [Online]. Available: https://magenta.tensorflow.org/2016/07/15/lookback-rnn-attention-rnn

[18] B. Felbo, A. Mislove, A. Søgaard, I. Rahwan, and S. Lehmann, "Using millions of emoji occurrences to learn any-domain representations for detecting sentiment, emotion and sarcasm," *arXiv preprint arXiv:1708.00524*, 2017.

[19] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[20] Z. Yang, D. Yang, C. Dyer, X. He, A. Smola, and E. Hovy, "Hierarchical attention networks for document classification," in *Proceedings of the 2016 conference of the north american chapter of the association for computational linguistics: Human language technologies*, 2016, pp. 1480–1489.

[21] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[22] J. Perricone, *Melody in songwriting: Tools and techniques for writing hit songs*. Hal Leonard Corporation, 2000.

[23] J.-P. Briot and F. Pachet, "Music generation by deep learning - challenges and directions," *Neural Computing and Applications*, Oct. 2018 [Online]. Available: http://arxiv.org/abs/1712.04371. [Accessed: 20-Mar-2019]

[24] D. Müllensiefen and M. Pendzich, "Court decisions on music plagiarism and the predictive value of similarity algorithms," *Musicae Scientiae*, vol. 13, no. 1, pp. 257–295, Mar. 2009 [Online]. Available: http://journals.sagepub.com/doi/10.1177/102986490901300111. [Accessed: 20-Mar-2019]

[25] A. Papadopoulos, P. Roy, and F. Pachet, "Assisted lead sheet composition using flowcomposer," in *International conference on principles and practice of constraint programming*, 2016, pp. 769–785.

[26] A. Fan, M. Lewis, and Y. Dauphin, "Hierarchical neural story generation," *CoRR*, vol. abs/1805.04833, 2018 [Online]. Available: http://arxiv.org/abs/1805.04833

[27] M. T. Wong, A. H. W. Chun, Q. Li, S. Chen, and A. Xu, "Automatic haiku generation using vsm," in *WSEAS international conference. Proceedings. Mathematics and computers in science and engineering*, 2008.

[28] Y.-w. Guo, J.-h. Yu, X.-d. Xu, J. Wang, and Q.-s. Peng, "Example based painting generation," *Journal of Zhejiang University-Science A*, vol. 7, no. 7, pp. 1152–1159, 2006.

[29] D. E. Rumelhart, G. E. Hinton, R. J. Williams, and others, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 5, no. 3, p. 1, 1988.

[30] H. H. Mao, T. Shin, and G. W. Cottrell, "DeepJ: Style-specific music generation," *2018 IEEE 12th International Conference on Semantic Computing (ICSC)*, pp. 377–382, Jan. 2018 [Online]. Available: http://arxiv.org/abs/1801.00887. [Accessed: 20-Mar-2019]

[31] A. Nayebi and M. Vitelli, "GRUV: Algorithmic music generation using recurrent neural networks," *Course CS224D: Deep Learning for Natural Language Processing (Stanford)*, 2015.

[32] A. Karpathy, "The unreasonable effectiveness of recurrent neural networks." [Online]. Available: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

[33] C. Sammut and G. I. Webb, *Encyclopedia of machine learning and data mining*. Springer Publishing Company, Incorporated, 2017.

[34] S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber, and others, "Gradient flow in recurrent nets: The difficulty of learning long-term dependencies." A field guide to dynamical recurrent neural networks. IEEE Press, 2001.

[35] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[36] M. De Coster, "Polyphonic music generation with style transitions using recurrent neural networks," 2017 [Online]. Available: https://lib.ugent.be/fulltxt/RUG01/002/367/003/RUG01-002367003_2017_0001_AC.pdf. [Accessed: 20-Mar-2019]

[37] D. Eck and J. Schmidhuber, "A first look at music composition using LSTM recurrent neural networks," p. 11.

[38] A. Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, 2013.

[39] O. Melamud, J. Goldberger, and I. Dagan, "Context2vec: Learning generic context embedding with bidirectional lstm," in *Proceedings of the 20th signll conference on computational natural language learning*, 2016, pp. 51–61.

[40] L. Weng, "Attention? Attention!" [Online]. Available: https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html. [Accessed: 20-Mar-2019]

[41] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[42] K. Cho *et al.*, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[43] B. L. Sturm, J. F. Santos, O. Ben-Tal, and I. Korshunova, "Music transcription modelling and composition using deep learning." 2016 [Online]. Available: http://arxiv.org/abs/1604.08723

[44] C. Walshaw, "Abc notation home page." [Online]. Available: http://abcnotation.com/. [Accessed: 14-Apr-2019]

[45] D. D. Johnson, "Generating polyphonic music using tied parallel networks," in *International conference on evolutionary and biologically inspired music and art*, 2017, pp. 128–143.

[46] K. Choi, G. Fazekas, and M. Sandler, "Text-based lstm networks for automatic music composition," *arXiv preprint arXiv:1604.05358*, 2016.

[47] A. Huang and R. Wu, "Deep learning for music," *arXiv preprint arXiv:1606.04930*, 2016.

[48] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[49] L. van der Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[50] G. Hadjeres and F. Nielsen, "Interactive music generation with positional constraints using anticipation-RNNs," *arXiv:1709.06404 [cs, stat]*, Sep. 2017 [Online]. Available: http://arxiv.org/abs/1709.06404. [Accessed: 20-Mar-2019]

[51] I. Simon, A. Roberts, C. Raffel, J. Engel, C. Hawthorne, and D. Eck, "Learning a latent space of multitrack measures," Jun. 2018 [Online]. Available: https://arxiv.org/abs/1806.00195v1. [Accessed: 12-Mar-2019]

[52] A. Elgammal, B. Liu, M. Elhoseiny, and M. Mazzone, "Can: Creative adversarial networks, generating" art" by learning about styles and deviating from style norms," *arXiv preprint arXiv:1706.07068*, 2017.

[53] I. Goodfellow *et al.*, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[54] M. A. Casey, R. Veltkamp, M. Goto, M. Leman, C. Rhodes, and M. Slaney, "Content-based music information retrieval: Current directions and future challenges," *Proceedings of the IEEE*, vol. 96, no. 4, pp. 668–696, 2008.

[55] R. Typke, P. Giannopoulos, R. C. Veltkamp, F. Wiering, and R. Van Oostrum, "Using transportation distances for measuring melodic similarity," 2003.

[56] R. Typke, R. C. Veltkamp, and F. Wiering, "Searching notated polyphonic music using transportation distances," in *Proceedings of the 12th annual acm international conference on multimedia*, 2004, pp. 128–135.

[57] I. Sutskever, J. Martens, and G. E. Hinton, "Generating text with recurrent neural networks," in *Proceedings of the 28th international conference on machine learning (icml-11)*, 2011, pp. 1017–1024.

[58] D. Eck and J. Schmidhuber, "Finding temporal structure in music: Blues improvisation with lstm recurrent networks," in *Proceedings of the 12th ieee workshop on neural networks for signal processing*, 2002, pp. 747–756.

[59] Y. Gal and Z. Ghahramani, "A theoretically grounded application of dropout in recurrent neural networks," in *Advances in neural information processing systems*, 2016, pp. 1019–1027.

[60] S. Dong, "SAM-bach: A deep generative model for bach chorale generation," PhD thesis, McGill University Libraries, 2018.

[61] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[62] H. Tieleman T., "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural Networks for Machine Learning 4*.

[63] K. Xu *et al.*, "Show, attend and tell: Neural image caption generation with visual attention," in *International conference on machine learning*, 2015, pp. 2048–2057.

[64] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[65] A. Karpathy, "A peek at trends in machine learning." [Online]. Available: https://medium.com/@karpathy/a-peek-at-trends-in-machine-learning-ab8a1085a106

[66] G. Ewer, "Writing a song backwards: Yes, it works" [Online]. Available: https://www.secretsofsongwriting.com/2012/02/21/writing-a-song-backwards-yes-it-works/

[67] O. Mogren, "C-rnn-gan: Continuous recurrent neural networks with adversarial training," *arXiv preprint arXiv:1611.09904*, 2016.

[68] D. N. Baker, "Improvisation: A tool for music learning," *Music Educators Journal*, vol. 66, no. 5, pp. 42–51, 1980.

[69] C. McKay and I. Fujinaga, "JSymbolic: A feature extractor for midi files." in *ICMC*, 2006.

[70] C.-Z. A. Huang *et al.*, "Music transformer," *arXiv:1809.04281 [cs, eess, stat]*, Sep. 2018 [Online]. Available: http://arxiv.org/abs/1809.04281. [Accessed: 20-Mar-2019]

[71] A. Ycart, E. Benetos, and others, "A study on lstm networks for polyphonic music sequence modelling," 2017.

[72] N. Mauthes, "VGM-rnn: Recurrent neural networks for video game music generation," 2018.

[73] H.-W. Dong, W.-Y. Hsiao, L.-C. Yang, and Y.-H. Yang, "MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment," in *Thirty-second aaai conference on artificial intelligence*, 2018.

[74] N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent, "Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription," *arXiv preprint arXiv:1206.6392*, 2012.

[75] S. Lattner, M. Grachten, and G. Widmer, "Imposing higher-level structure in polyphonic music generation using convolutional restricted boltzmann machines and constraints," *arXiv preprint arXiv:1612.04742*, 2016.

[76] C. Raffel and D. P. Ellis, "Intuitive analysis, creation and manipulation of midi data with pretty_midi," in *15th international society for music information retrieval conference late breaking and demo papers*, 2014, pp. 84–93.

[77] H.-W. Dong, W.-Y. Hsiao, and Y.-H. Yang, "Pypianoroll: Open source python package for handling multitrack pianoroll," *Proc. ISMIR. Late-breaking paper;[Online] https://github.com/salu133445/pypianoroll*, 2018.

[78] P. M. Todd, "A connectionist approach to algorithmic composition," *Computer Music Journal*, vol. 13, no. 4, pp. 27–43, 1989.

[79] C. A. Huang *et al.*, "An improved relative self-attention mechanism for transformer with application to music generation," *CoRR*, vol. abs/1809.04281, 2018 [Online]. Available: http://arxiv.org/abs/1809.04281

[80] F. Chollet and others, "Keras: Deep learning library for theano and tensorflow," *URL: https://keras.io/k*, vol. 7, no. 8, p. T1, 2015.

[81] I. J. Goodfellow, "NIPS 2016 tutorial: Generative adversarial networks," *CoRR*, vol. abs/1701.00160, 2017 [Online]. Available: http://arxiv.org/abs/1701.00160

[82] S. Dalla Bella, V. B. Penhune, N. Kraus, K. Overy, C. Pantev, and J. S. Snyder, *The neurosciences and music iii: Disorders and plasticity*, vol. 1169. John Wiley & Sons, 2009.

[83] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan," *arXiv preprint arXiv:1701.07875*, 2017.

[84] F. Farnstrom, J. Lewis, and C. Elkan, "Scalability for clustering algorithms revisited," *SIGKDD explorations*, vol. 2, no. 1, pp. 51–57, 2000.

[85] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain." *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[86] S. Haykin, *Neural networks: A comprehensive foundation*. Prentice Hall PTR, 1994.

[87] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[88] K. Jarrett, K. Kavukcuoglu, Y. LeCun, and others, "What is the best multi-stage architecture for object recognition?" in *2009 ieee 12th international conference on computer vision*, 2009, pp. 2146–2153.

[89] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014 [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html

[90] C. Ammer, *The facts on file dictionary of music*. Infobase Publishing, 2004.

[91] E. W. Marvin, "A generalization of contour theory to diverse musical spaces: Analytical applications to the music of dallapiccola and stockhausen," *Concert music, rock, and jazz since*, pp. 135–171, 1945.

[92] P. Roy, A. Papadopoulos, and F. Pachet, "Sampling variations of lead sheets," *arXiv:1703.00760 [cs]*, Mar. 2017 [Online]. Available: http://arxiv.org/abs/1703.00760. [Accessed: 12-Mar-2019]

[93] G. Brunner, Y. Wang, R. Wattenhofer, and J. Wiesendanger, "JamBot: Music theory aware chord based generation of polyphonic music with LSTMs," in *2017 IEEE 29th international conference on tools with artificial intelligence (ICTAI)*, 2017, pp. 519–526 [Online]. Available: https://ieeexplore.ieee.org/document/8371988/. [Accessed: 12-Mar-2019]

[94] L. Casini and M. Roccetti, "The impact of AI on the musical world: Will musicians be obsolete?" *Studi di estetica*, vol. 0, no. 12, Dec. 2018 [Online]. Available: http://mimesisedizioni.it/journals/index.php/studi-di-estetica/article/view/630. [Accessed: 12-Mar-2019]

[95] G. Brunner, A. Konrad, Y. Wang, and R. Wattenhofer, "MIDI-VAE: Modeling dynamics and instrumentation of music with applications to style transfer," *arXiv:1809.07600 [cs, eess, stat]*, Sep. 2018 [Online]. Available: http://arxiv.org/abs/1809.07600. [Accessed: 12-Mar-2019]

[96] J. Lee, S. Park, S. Jo, and C. D. Yoo, "Music plagiarism detection system," p. 3, 2011.

[97] S. Oore, I. Simon, S. Dieleman, D. Eck, and K. Simonyan, "This time with feeling: Learning expressive musical performance," *arXiv:1808.03715 [cs, eess]*, Aug. 2018 [Online]. Available: http://arxiv.org/abs/1808.03715. [Accessed: 20-Mar-2019]

[98] J. Alammar, "The illustrated transformer" [Online]. Available: https://jalammar.github.io/illustrated-transformer/

Appendix A: Introduction to Neural Networks

Machine Learning

Machine Learning is a field at the overlap between Computer Science and Statistics, being an area of Artificial Intelligence. It is characterized by having algorithms that are not clearly restricted to solving a specific task, but that are suitable for solving many different tasks. For example, an algorithm could be employed for both classifying images and classifying strings of text.

Supervised Machine Learning

In *supervised* machine learning, the software is programmed to learn the central features of a given set of data (also called the *training set*). The program is then shown new data (also called the *test set*) and asked to produce predictions on this new set. For example, the data could be sets of brain scans, and the outputs could be a boolean value representing whether the brain scan is showing signs of a particular disorder.

In these cases, the dataset is described as a set of examples along with corresponding labels.

Basic examples of these are regression and classification, where the model is learning $P(y|x, \theta)$, the probability of a label y given the example x and the model θ. This represents the conditional density estimation.

In the linear regression model every data point x_i can be understood as a combination that yields the label y. The label, thus, can be obtained by a linear calculation of $x_1, ...x_n$. This can be formally defined by this equation:

$$y(x) = \sum_{i=1}^{m} w_i \cdot x_i + b = w^T x + b$$

In this b is the bias term, or residual error.

If we apply a logistic function to the linear combination we obtain the logistic regression. This allows us to map a value of the calculation in the 0 - 1 range. Thus we can obtain a classification model, where 0 and 1 represent an exclusive binary class (e.g. healthy and not healthy, in the medical field). The name of this model is *logistic regression*, after the logistic (sigmoid) function.

The sigmoid function in question is defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{6.1}$$

This forms the basis of the activation functions that I define in their own section

Unsupervised Machine Learning

In its *unsupervised* form, the algorithms are not asked to predict a specific label. Rather, they simply learn the features of the data. These are then employed in multiple ways: clustering, dimensionality reduction.

One example of unsupervised learning is clustering, where the objective is to separate the data points into separate classes or partitions [33], based on various distance measures. One of the most common algorithms for clustering is *K-means* [84]. The approach is to initiate with an imperfect configuration, reassign the examples, and then recalculate the centers until a certain convergence criterion is met [33].

Deep Learning

In this section, I will discuss and present the relevant topics in the field of machine learning, and, especially deep learning, that form the base for my own investigations. I argue for my choice of these specific techniques and how they can be employed in the music generation problem in the chapter on Methods.

Multi-layer perceptron

Deep learning gets its name from the multiple layers of learners stacked on top of each other. This is most evident in the multilayer perceptron, the basic form of deep learning. It is also called a feed-forward network, as inputs are processed by every layer and propagated forward to the next layers.

The multilayer perceptron is formed, as the name entails, of multiple perceptrons. The perceptron [85] is a simple binary classifier learning algorithm. The limitation inherent in the perceptron is that can only correctly predict a space of data that is linearly separable [86]. *Linear separability* describes the set of classification problems where the data can be perfectly separated with a single hyperplane. Consider the data in fig. 6.1. We can see that the two classes (circles and X-es) are perfectly separated by the dotted line.

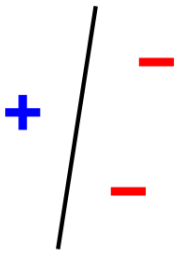


Figure 6.1: Linear separability of a simple dataset[1]

There are however plenty of cases where the data is not linearly separable. In such cases, the simple perceptron fails to perform. An example of this, as discussed in [87], is the logical XOR (exclusive OR) function. See fig. 6.2 for how a sample of the data would look like in a scatter plot, and tbl. 6.1 for the truth table of the operation. We can see that we require two hyperplanes in order to separate the classes.

[1] Image from https://commons.wikimedia.org/wiki/File:VC1.svg. Creative Commons Attribution-Share Alike 3.0 License.

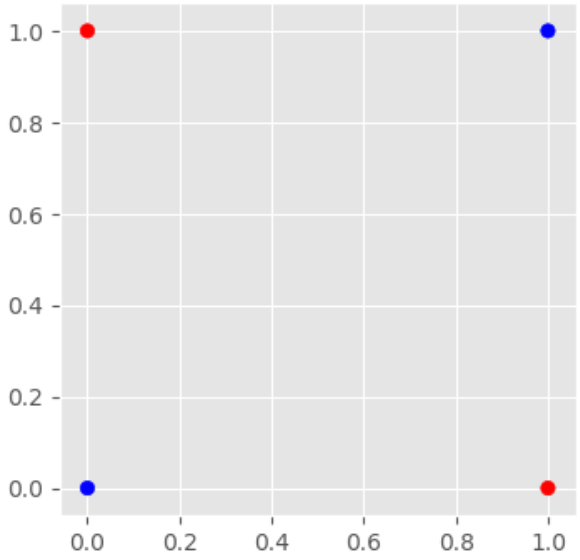


Figure 6.2: Example of linearly inseparable data: the XOR function

Table 6.1: XOR table

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

In order to solve such a problem, multiple perceptrons can be combined together to approximate the required function. We will then have multiple perceptrons each modelling one decision surface. A hierarchy allows perceptrons higher in the order to learn sub-classification problems part of the bigger classification task.

In a multi-layer perceptron, multiple such learners are added together to form a more complex learner system. They are usually defined as having one input layer, one or more hidden layers, and an output layer. The input layer processed the input data directly. The hidden layers process the output of the input layer and provide added complexity to learning pipeline. Finally, the output layer outputs a final probability distribution over the target classes [87]. These are the "quintessential deep learning models" [87, p. 163]. They are trained to learn some function g that would describe the relation between the input x and the labels y. "A feedforward network defines a mapping $y = g(x; \theta)$ and learns the value of the parameters θ that result in the best function approximation" [87, p. 163].

A i-th neuron in hidden layer $n+1$ can be formally represented as:

$$h_{h+1,i} = f(w_{n,i}^T h_n + b_{n+1,i})$$

where:

- f is the activation function
- h_n is the output of the previous n-th layer

- $w_{n,i}$ are the weights connecting the n layer outputs to the i layer
- $b_{n+1,i}$ is the bias term

The final layer will usually output the probabilities after the application of a non-linear activation function, like softmax or sigmoid.

In fig. 6.3 we can see a graphical representation of MLP with 3 input notes (red), one hidden layer of 4 perceptrons (olive), and an output layer with 1 neuron (green).

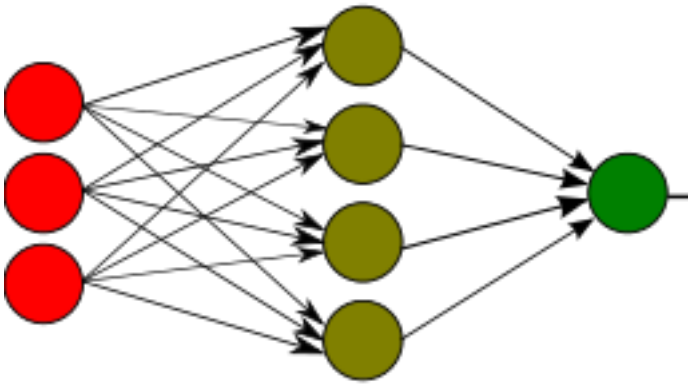


Figure 6.3: Multi layer perceptron[2]

A drawback of this model is that the feedforward networks do not have any feedback mechanism that would allow the system's outputs to be fed back into itself. This means that at every step the network is oblivious of where in the sequence of the data it is. As I will discuss in the section on RNNs and LSTMs, this proves essential in learning the patterns in sequential and temporal data (like stock exchange prediction and, in my case, music generation).

Activation functions Activation functions are non-linear functions that are applied to the output of a cell (perceptron). These allow the model to learn functions that are not strictly a linear combination of the input values. This is essential to learning complex patterns in data.

Formally, an activation function g is applied as follows:

$$y = g(x^T W + b)$$

The two functions I have already mentioned, sigmoid and softmax, are basically the same function, but applied to binary classification, and multi-class classification, respectively. While the sigmoid function is defined in eq. 6.1, I will define the softmax function here:

$$softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

We pass each element in the array through the exponential function e, then divide it by the sum of all the other elements passed through the same exponential function.

[2]Image from https://commons.wikimedia.org/wiki/File:MultiLayerPerceptron.svg. Creative Commons Attribution-Share Alike 3.0 Unported License.

One of the most commonly employed activation functions, apart from the already mentioned softmax and sigmoid, is the rectified linear unit (or ReLU) function [88], defined as $g(z) = max\{0, z\}$. The advantage of this activation function is that it can capture both non-linearity and linearity at the same time: "because rectified linear units are nearly linear, they preserve many of the properties that make linear models easy to optimize with gradient-based methods. They also preserve many of the properties that make linear models generalize well" [87, p. 169].

Gradient descent learning

A single pass of the inputs through the multi-layer perceptron will only produce output from the neural network. The problem is then: how do we optimize the weights of the network (the w parameters of each of the perceptrons that form the overall model θ)? Gradient descent is a method of searching for an optimal set of weights θ by using a measure of the error of the model.

Loss functions

These measures are called loss functions. They are each used for different tasks. The most common ones are Root Mean Squared Error (RMSE) and Cross Entropy (CE). The former is common in regression problems, while the latter is mostly employed in classification problems. RMSE is defined as follows:

$$\sqrt{\frac{\sum_{i=1}^{n}(\hat{y} - y)^2}{n}}$$

Where:

- n is the number of elements in the data set
- $\hat{y}$ is the correct label of the example i
- y is the label predicted by the network

It basically measures the absolute distance between the predicted scalar and the expected answer. By using the squared root function it discourages large errors.

Cross Entropy is defined as follows:

$$-\sum_{c=1}^{M} y_{i,c} \log(p_{i,c})$$

Where:

- M is the number of distinct classes
- p is the probability in the range 0 - 1 that the i observation is in the class c

By using the *log* function the Cross Entropy function penalizes those errors that are also extremely confident.

Backpropagation

After calculating the loss for a given training sample, we update every weight parameter ($w_{n,i}$, the weight connecting n to i). This is done by obtaining "the negative gradient of the error measure with respect to that parameter" [33, p. 97]:

$$\triangle w_{n,i} = -\alpha \cdot \frac{\partial W}{\partial w_{n,i}}$$

Where:

- α is the step size, or learning rate. This parameter controls the amount by which the weights change when updating them. When this is a small number, the system converges smoothly, but at a sluggish pace. When it is a large number, the system might converge faster, at the risk of overstepping the optimum;
- ∂ represents the partial derivate;

In essence, we want to minimize the loss function (also called objective function, cost function, or error function [87, p. 80]). Formally, we use the derivative ($f'(x)$, or $\frac{dy}{dx}$) in order to compute the gradient of the function. We then take "small steps with the opposite sign of the derivative" [87, p. 80]. We go in the opposite direction because we want to minimize the loss function, not maximize it. This technique is called gradient descent. The process can be visualized in fig. 6.4.

Figure 6.4: Gradient descent process[3]

In order to speed up the learning process, the model training can be done in batches, instead of at the rate of an individual sample. In this way, we accumulate the $\triangle$ values over multiple samples. When we reach a certain number of samples processed, the signal is propagated backwards into the network to update the weights w.

The main difficulty faced in optimizing a model using backpropagation is the issue of local optima, an "inherent problem of gradient descent" [33, p. 100]. Instead of reaching the global optimum of the model, it is possible that the model might end up confined to a region that is bounded by a higher ground (also called critical points or stationary points [87]). This can be formally defined as moments when the derivative of the function is zero ($f'(x) = 0$). At these points, the derivative does not offer any direction into which to continue. In order to alleviate these problems, different approaches have been proposed throughout the field. In the following, I will discuss those that are relevant to my research project.

[3]Image based on image from https://www.neural-networks.io/en/single-layer/gradient-descent.php

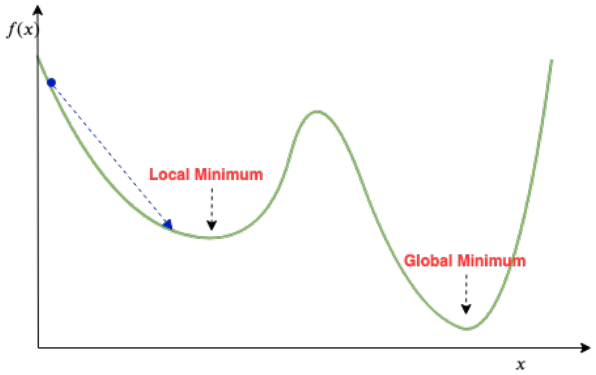


Figure 6.5: Local minimum vs global minimum[4]. The main challenge in gradient descent methods is reaching a local minimum instead of a global minimum. This can happen because at these points the gradient does not provide any further directions into which to advance

Optimization techniques for gradient descent

One of the problems facing neural networks is overfitting, where the model begins to learn to mimic the noise in the data, rather than the meaningful underlying patterns [33, p. 944]. It can be seen as a facet of the bias-variance trade-off. As the model complexity and the number of parameters increase, the model risks overfitting the data. A symptom of this is a decrease in the error rate on the training set but with an increase in the error rate on the validation set.

One solution to this problem comes with the **dropout** mechanism [89] This randomly deactivates a percentage of the cells during each training epoch in order to force the network to adapt and to better balance the learning.

Another problem commonly encountered in the field is stagnation, where the system stops learning. This is sometimes due to a fixed learning rate. This means that the gradient descent method is simply jumping from one slope to another, not being able to descend. In fact, this can even cause the step function to diverge (see fig. 6.6).

[4]Image based on image from https://www.neural-networks.io/en/single-layer/gradient-descent.php

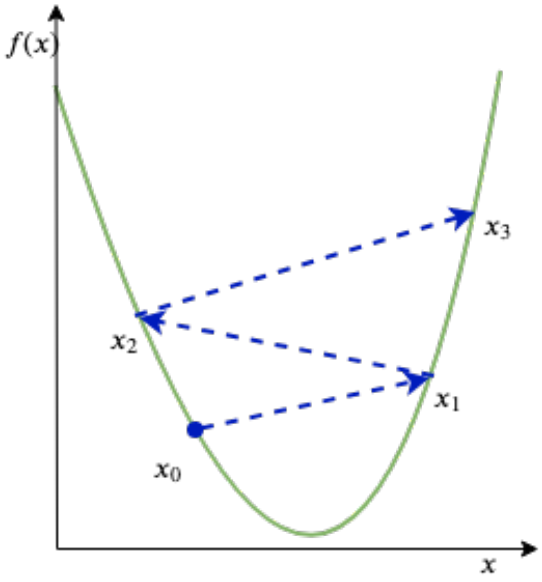


Figure 6.6: Gradient descent diverges from the minimum due to a too large learning rate[5]

One proposed solution to this is to control the learning rate when training reaches a plateau. For example, if the network's loss does not decrease over a certain number of epochs, we decrease the learning rate. The Keras deep learning library [80] allows for the user to use this functionality.

[5]Image based on image from https://towardsdatascience.com/gradient-descent-in-a-nutshell-eaf8c18212f0

Appendix B: Music theory

While this thesis does deal with topic related to music theory and music writing, it is not my focus to provide an in-depth description of the history, practice, and meaning of the musical concepts mentioned. I will attempt however a short introductory overview of the main ideas that will be relevant to the discussion. Such an introduction can help the reader understand the reasoning behind the decisions I will make in this project.

The most common terms in music theory are pitch, rhythm, and harmony. These are the building blocks for what constitutes a musical piece. I base most of these explanations on [90].

Pitch can be defined in two ways, depending on the perspective. From an acoustics angle, it is the frequency at which a given note vibrates, "the number of vibrations per second of the string, air column, or other sound-producing agent" [90, p. 316]. This is represented in Hertz (Hz). From a strictly musical perspective, it is the note that the instrument is playing (e.g. C3). The C3 notation can be read as playing the note C (or Do, in Solfege notation) in the third octave. This corresponds to 130.81 Hz, and 48 in the MIDI range (see section below for more about MIDI).

Closely related to pitch is the term "pitch class". This simply represents all the notes of the same note name (e.g. A, or C), irrespective of the octave it is played in [90, p. 316]. This is analyzed in the section of objective evaluation, in order to obtain a quantitative analysis of the datasets, and in experiments, in order to compare the performance of the models. There are twelve pitch classes: A, A#/Bb, B, C, C#/Db, D, D#/Eb, E, F, F#/Gb, G, G#/Ab..[6]

Rhythm is the overall combination of tempo (the speed at which the notes are played, in beats per minutes, BPM) and the overall contrasts between short and long notes [90, p. 348]. Another important part of rhythm is the concept of time signature, which represents how a given unit of musical time (a bar) is split [90, p. 431]. Common divisions for time are: a bar, a beat (or quarter, in relation to the bar), an eighth (again, in relation to the bar), and a sixteenth (same). In fig. 6.7 we can see how these subdivisions add up to form a bar. A bar can consist of either 4 quarter notes, 8 eights, 16 sixteenths, or any combinations of these. It is important to note that this only holds for a time signature of 4/4.

[6] The reason we have both sharps (#) and flats (b) is due to a limitation to writing sheet music. In writing sheet music there can only be one pitch class of a given letter in a scale. Thus, if a scale has, e.g. A and A#, the A# needs to be changed to a Bb in order to accommodate the stave. See https://music.stackexchange.com/questions/67046/why-are-there-both-sharps-and-flats

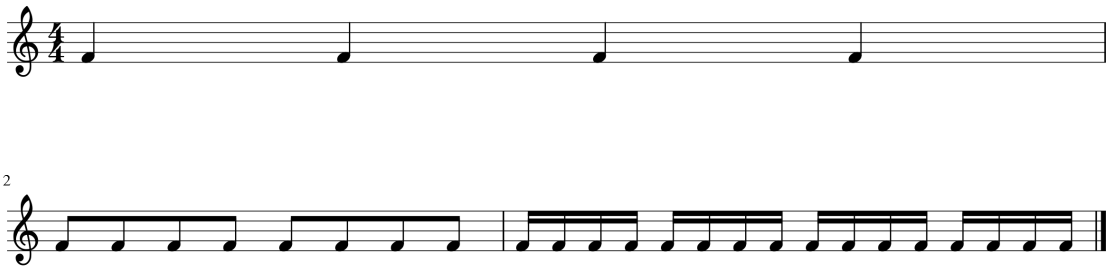


Figure 6.7: Subdivisions

Harmony is defined as the overall combination of multiple notes being played at the same simultaneous time. The notes form an overall harmonic effect upon the listener. This can be consonant (pleasing) or dissonant (tense), in accordance with rules related to their psychoacoustic properties and how their frequencies relate to each other. Since these notes occur concomitantly harmony is also seen as 'the "vertical" aspect of music' [90, p. 176]. In linear algebra terms, it can be seen as the Y axis. While this is rather simplistic, as "harmony also involves horizontal movement from one chord to another" [90, p. 176], is helps solidify the essential difference between it and melody.

Within harmony we also have the concept of "chromaticism". This refers to using the full extent of all the twelve pitch classes, instead of restricting to a given scale (which is referred to as by "diatonicity"). These are relevant to the current work. Ideally, I want my system to favour diatonicity in order to obtain harmonically-pleasing melodies. There are, of course, genres of music where chromaticism is employed to great effect: jazz, metal, avant-garde contemporary classical etc.

Melody, as contrasted with harmony, is the "horizontal" domain of musical experience, or the X axis. It concerns with the temporal development of a musical idea or tune. It can be defined as a chronological ordering of pitches that is central to a musical piece. The rules by which melodies are composed have varied throughout time and are usually centred around a choice of a specific scale.

Melodic contour is essential to understanding the emotional and aesthetic impact of a musical piece. Elizabeth West Marvin provides a discussion of melodic contour in [91] (as collected in [2]). She quotes Arthur Schonberg's view on the wave-like shape of these: "melodies proceed in waves, a fact which can readily be observed here. The amplitude of these waves varies. A melody seldom moves long in one direction". She also mentions Ernst Toch's categorization efforts of contours. The author provides examples of sheet music with illustrations of the sine-like shape of the melodies. While the author concludes that such analyses do not uncover much about how different pitches work together within a melody, they can "reveal much about a work's underlying structure" (p.171). This relates to the view employed by the MelodyShape algorithm [14], where the author employs a geometric model to encode musical pieces with "curves in the pitch-time plane" [15, p. 344].

A scale is a collection of pitch classes arranged in rising order. In Western traditions, a scale is usually a diatonic scale. This means that there are eight pitch classes selected from the chromatic scale (all the possible 12 pitch classes).

An octave represents a cycle of a given scale (e.g. A1 - A2). Notes an octave apart sound similar because the higher note vibrates at exactly double the frequency of the lower register note [90, p. 267]

MIDI

MIDI was developed in the mid-1980s, as a protocol for new audio equipment. This allowed for easy communication between the different gear you could find in the studio or on stage. With the format musicians and sound engineers could connect different pieces of equipment together. MIDI allows for a diverse range of instructions and messages. A very common approach in electronic music-making is to connect a MIDI keyboard (that does not produce any sound of its own) to a computer, perform a musical piece, and have a computer software (Digital Audio Workstation, or DAW, for short) record that in symbolic form. This sequence can then be manipulated, both at a symbolic level, by changing the duration, pitch, or velocity of a note, or at the audio level, by applying different effects on the synthesized audio. In order for the MIDI sequence to become audio data, a synthesizer software is required.

In MIDI encoding, notes are triggered by NOTE ON events, and stopped by NOTE OFF events. MIDI notes are also defined by their velocity, which represents how loud the notes should be played by the synth software. All of these attributes are in the 0-127 range (128 possible values). A standard MIDI keyboard, with 88 keys, corresponds to the range of 21 (A0, in pitch notation) - 108 (C8) in the MIDI range.

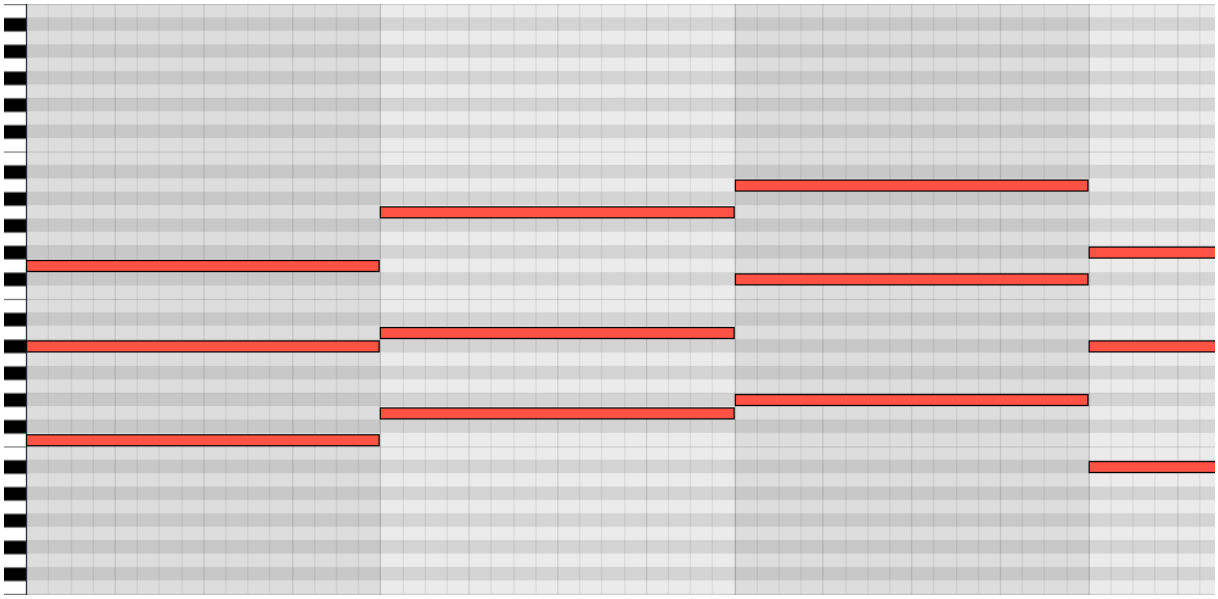


Figure 6.8: Pianoroll example of a sequence of chords. From the DAW Ableton Live

A common representation of such a MIDI sequence in a DAW is called a pianoroll. Such a representation, as can be seen in fig. 6.8, is a bidimensional representation, with the X axis as time, and the Y axis as pitch. This translates well to a binary matrix format, P, where $P_{i,j}$ being 0 or 1 represents whether the note at time step i and pitch j is played. In fact, as I will discuss in the Dataset section, this is indeed the most common format used in research.